RL-TR-96-118
Final Technical Report
July 1996

SIMS: SINGLE INTERFACE TO MULTIPLE SOURCES

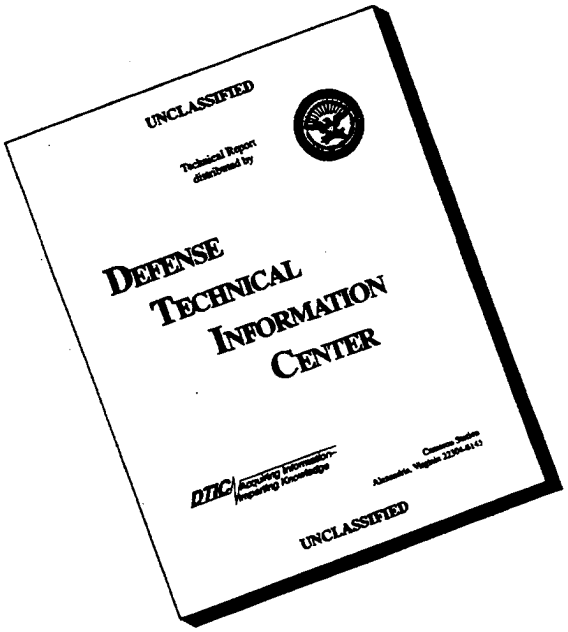
University of Southern California

Sponsored by
Advanced Research Projects Agency

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Rome, New York

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be releasable to the general public, including foreign nations.

RL-TR-96-118 has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Division

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify Rome Laboratory/ (C3CA), Rome NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

SIMS: SINGLE INTERFACE TO MULTIPLE SOURCES

Yigal Arens
Craig A. Knobloch
Chin Y. Chee
Chunnan Hsu

Contractor: University of Southern California
Contract Number: F30602-91-C-0081
Effective Date of Contract: 08 July 1991
Contract Expiration Date: 02 February 1995
Short Title of Work: SIMS: Single Interface to Multiple Sources
Period of Work Covered: Jul 91 - Mar 95

Principal Investigator: Yigal Arens
Phone: (310) 822-1511

RL Project Engineer: Raymond A. Liuzzi
Phone: (315) 330-3528

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by Raymond A. Liuzzi, RL/C3CA, 525 Brooks Rd, Rome NY 13441-4505.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1996	Final Jul 91 - Dec 95

4. TITLE AND SUBTITLE

SIMS: SINGLE INTERFACE TO MULTIPLE SOURCES

5. FUNDING NUMBERS

C - F30602-91-C-0081
PE - 62301E
PR - 5581
TA - 20
WU - 46

6. AUTHOR(S)

Yigal Arens, Craig A. Knobloch, Chin Y. Chee, and Chunnan Hsu

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Southern California
Information Sciences Institute
4676 Admiralty Way
Marina Del Ray CA 90292

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Advanced Research Projects Agency
3701 North Fairfax Drive
Fairfax VA 22203-1714

Rome Laboratory (C3CA)
525 Brooks Rd
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

RL-TR-96-118

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Raymond A. Liuzzi/C3CA/(315)330-3528

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

With the current explosion of data, retrieving and integrating information from various sources is a critical problem. This report describes work performed at USC/ISI, aimed at developing a general and extensible approach to this problem. The SIMS approach exploits a semantic model of a problem domain to integrate the information from various sources, i.e., databases and knowledge bases. The domain and the information sources are modeled. Queries submitted to SIMS are mapped into a set of queries to individual information sources. The set of queries is then further optimized, using knowledge about the domain and the information sources. The data obtained is then returned to the user. SIMS utilizes techniques from the areas of knowledge representation, planning, and learning.

14. SUBJECT TERMS

Distributed databases, Learning, Multidatabase, Planning, SIMS

15. NUMBER OF PAGES

140

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Contents

Abstract

With the current explosion of data, retrieving and integrating information from various sources is a critical problem. This report describes work performed at USC/ISI, aimed at developing a general and extensible approach to this problem. The SIMS approach exploits a semantic model of a problem domain to integrate the information from various sources — databases and knowledge bases. The domain and the information sources are modeled. Queries submitted to SIMS are mapped into a set of queries to individual information sources. The set of queries is then further optimized, using knowledge about the domain and the information sources. The data obtained is then returned to the user. SIMS utilizes techniques from the areas of knowledge representation, planning. and learning.

This report is structured as follows. Chapter 1 is devoted to overviews, first of the technological infrastructure used by SIMS, and then of the operation of the SIMS system itself. Chapter 2 follows with a more detailed description of the SIMS system. Subsequent Chapters provide more details about various specific aspects of SIMS' operation. Specifically, Chapter 3 describes SIMS' planner, Chapter 4 describes SIMS' semantic query optimization techniques, and Chapter 5 describes the learning process used to obtain query optimization rules. We conclude with an appendix, a tutorial describing how to set up a SIMS system to access new information sources.

Authorship of this report: this report was written by Yigal Arens, Chin Y. Chee, Chunnan Hsu, and Craig A. Knoblock. Chunnan Hsu's work as a Graduate Research Assistant was supported in part by National Science Foundation grant No. IRI-9313993.

Chapter 1

Overview and Summary of SIMS

1 Introduction

Most tasks performed by users of complex information systems involve interaction with multiple information sources.[1] Examples can be found in the areas of analysis (e.g., of intelligence data or logistics forecasting) and in resource planning and briefing applications. Retrieval of desired information dispersed among multiple sources requires general familiarity with their contents and structure, with their query languages, with their location on existing networks, and more. The user must break down a given retrieval task into a sequence of actual queries to information sources, and must handle the temporary storing and possible transformation of intermediate results — all this while satisfying constraints on reliability of the results and the cost of the retrieval process. With a large number of information sources, it is difficult to find individuals who possess the required knowledge, and automation becomes a necessity.

SIMS[2] accepts queries in the form of a description of a class of objects about which information is desired. This description is composed of statements in the Loom knowledge representation language (Section 1.1). The user is not presumed to know how information is distributed over the data- and knowledge bases to which SIMS has access — but he/she *is* assumed to be familiar with the application domain, and to use standard terminology to compose the Loom query. The interface enables the user to inspect the domain model as an aid to composing queries. SIMS proceeds to reformulate the user's query as a collection of more elementary statements that refer to data stored in available information sources. SIMS then creates a plan for retrieving the desired information, establishing the order and content of the various plan steps/subqueries. Using knowledge about the contents and structure of information sources, SIMS reformulates the plan to minimize its expected execution time. The resulting plan is then executed by performing local data manipulation and/or passing subqueries to the LIM system (Section 1.1), which generates the final translation into database queries in the appropriate language(s). A graphical user interface enables the user to inspect the plan in its various stages and to supervise its execution.

The SIMS project applies a variety of techniques and systems from Artificial Intelligence to build an intelligent interface to information sources. SIMS builds on the following ideas:

Knowledge Representation/Modeling, which is used to describe the domain about which information is stored in the information sources, as well the structure and contents of the information sources themselves. The domain model is a declarative description of the objects and activities possible in the application domain as viewed by a typical user. The model of each information source indicates the data-model used, query language, network location, size estimates, update frequency, etc., and describes the contents of its fields in terms of the domain model. The user formulates queries using terms from the application domain, without needing to know anything about specific information sources. SIMS' models of different information sources

---

[1] By the term *information source* we refer to any system from which information can be obtained. SIMS currently deals with Oracle databases and Loom knowledge bases.

[2] Services and Information Management for decision Systems. Subsequent to the period this report covers, the acronym was changed to stand for Single Interface to Multiple Sources.

are completely independent, greatly easing the process of incorporating new information sources into the system.

Planning/Search, which is used to construct a sequence of queries to individual information sources that will satisfy the user's query. A planner is used in an initial reformulation step that selects the information sources to be used in answering a query. It is also used to order the queries to the individual information sources, select the location for processing the intermediate data, and determine which queries can be executed in parallel.

Reformulation/Learning. SIMS considers alternative information sources and queries to them to retrieve the desired information. This search for more efficient query formulations is guided by the detailed semantics provided by the application domain model. Additional knowledge about the contents of the information sources may be learned from the databases and used to reformulate the queries.

An initial prototype incorporating many features of the SIMS approach has been built and applied to the domain of transportation planning — organizing the movement of personnel and materiel from one geographic location to another using available transportation facilities and vehicles [5]. An earlier prototype was applied to information needed for daily Naval briefings given in Hawaii about the status of the Pacific Fleet [3]. The system currently has access to nine Oracle databases and a Loom knowledge base with information about ships, ports, locations, relevant activities, etc. SIMS is controlled via a graphical user interface. It is written in Common Lisp and uses CLIM for its graphics.

There has been some work on the problem of accessing information distributed over multiple sources both in the AI-oriented database community and in the more traditional database community. Work in heterogeneous distributed databases includes the MULTIBASE, MERMAID, NDMS, IISS, IMDAS, ADDS, PRECI* and MRDSM systems. A survey and comparison of these can be found in [44]. Of these systems, only the first four attempt to support total integration of all information sources in the sense that SIMS provides. SIMS is distinguished from work in this community in that a complete semantic model of the application domain is created in a state-of-the-art knowledge representation language with powerful reasoning facilities. The model provides a collection of terms with which to describe the contents of (i.e., to create semantic models of) available information sources — and these include knowledge bases in addition to databases. Furthermore, a sophisticated planning mechanism is used at run-time in order to determine the potentially very complex relationship between the collection of information requested by the user and the data available from the various sources. In contrast to previous work, the domain model in SIMS is neither specific to a particular group of information sources, nor is there necessarily a direct mapping from the concepts in the model to the objects in the information sources. Our approach thus provides a much more flexible and easily extensible interface to a possibly changing collection of information sources.

The AI-oriented database community has done work on various aspects of using a knowledge base to integrate a variety of information sources. The Carnot project [16] integrates heterogeneous databases using a set of articulation axioms that describe how to map between SQL queries and domain concepts. Carnot uses the Cyc knowledge base [29] to build the articulation axioms, but after the axioms are built the domain model is no longer used or needed. In contrast, the domain model in SIMS is an integral part of the system, and allows SIMS to both combine information stored in the knowledge base and to reformulate queries. Illarramendi et al. [8, 24] present an approach to automatically integrating knowledge-base models from individual relational database schemas. In SIMS, the integration of the database models is not automated, although the translation of the individual database schemas into knowledge-base models is automated by the LIM system, which is used by SIMS. Elements of the approach described in that work can be applied to further automating the process of database modeling in SIMS. Finally, Papazoglou et al. [39] present a framework for intelligent information systems where, like SIMS, an explicit knowledge model is an integral part of an intelligent information agent.

Some additional related research has been performed by those working on semantic and object-oriented data models, e.g., [13, 23, 55]. Since they are interested in constructing a single DBMS, however, they take an almost diametrically opposed view of the problem from that of SIMS. While SIMS attempts to preserve its independence from the data models of the constituent data- and knowledge-bases, using a planner to bridge this gap at query time, they attempt to *closely integrate* the given data model into their DBMS.

```
(db-retrieve (?depth)
  (:and (port ?port)
        (port.name ?port "SAN-DIEGO")
        (port.depth ?port ?depth)))
```

Figure 1.1: Example SIMS/Loom Query

1.1 Technological Infrastructure

This subsection is provided for readers who may not be familiar with the systems underlying SIMS. A general understanding of Loom, LIM, and planners like Prodigy is assumed in the rest of this chapter.

Loom

Loom serves as the knowledge representation system SIMS uses to describe the domain model and the contents of the information sources, as well as serving as an information source in its own right. It provides both a language and an environment for constructing intelligent applications. Loom combines features of both frame-based and semantic network languages, and provides some reasoning facilities. As a knowledge representation language it is a descendent of the KL-ONE [9] system.

The heart of Loom is a powerful knowledge representation system, which is used to provide deductive support for the declarative portion of the Loom language. Declarative knowledge in Loom consists of definitions, rules, facts, and default rules. A deductive engine called a *classifier* utilizes forward-chaining, semantic unification and object-oriented truth maintenance technologies in order to compile the declarative knowledge into a network designed to efficiently support on-line deductive query processing. For a detailed description of Loom see [30, 31].

To illustrate both Loom and the form of SIMS' queries, consider Figure 1.1, which contains a simple semantic query to SIMS. This query requests the value of the depth of the San Diego port. The three subclauses of the query specify, respectively, that the variable `?port` describes a member of the model class `port`, that the relation `port.name` holds between the value of `?port` and the string `SAN-DIEGO`, and that the relation `port.depth` holds between the value of `?port` and the value of the variable `?depth`. The semantic query specifies that the value of the variable `?depth` be returned. A query to SIMS need not necessarily correspond to a single database query, since there may not exist one database that contains all the information requested.

LIM

In Loom the members of a class (e.g., the possible values of the variable `?port` in the expression in Figure 1.1) are *instances* in the knowledge base. In the case of large-sized realistic domains it is preferable not to define all objects of the domain as knowledge base instances. Instead, databases provide more efficient structures for organizing large numbers of such objects, and DBMSs are more efficient than AI languages for manipulating them.

The Loom Interface Module (LIM) [33] is being developed by researchers at Paramax Systems Corp. to mediate between Loom and databases. LIM reads an external database's schema and uses it to build a Loom representation of the database. The Loom user can then treat classes whose instances are stored in a database as though they contained "real" Loom instances. Given a Loom query for information in that class, LIM automatically generates a query in the appropriate database query language to the database that contains the information, and returns the results as though they were Loom instances. However, LIM focuses primarily on the issues involved in mapping a semantic query to a single database. After SIMS has planned a query and formed subqueries, each grounded in a single database, it hands the subqueries to LIM for the actual data retrieval. SIMS handles direct queries to the Loom knowledge base on its own.

Prodigy

The two problems of selecting information sources and ordering queries can be easily cast as planning problems. SIMS uses Prodigy [11, 37], a means-ends analysis planner, to solve both these problems. Prodigy

has an expressive operator and control language and has been linked to Loom, so that it can use the Loom domain model as its model of the world. SIMS formulates the selection of information sources and the ordering of queries as planning problems and hands them off to Prodigy.

A problem is specified in Prodigy by giving the system a set of operators that define the legal operations on a problem and an initial state description that defines the current state of the world. The system is then given a goal, which in this case is the query to be answered, and Prodigy generates a sequence of operators that transforms the initial state into a state in which the goal is satisfied.

Prodigy is used for solving the planning problems in SIMS for two main reasons. First, it provides an expressive language for both defining the problem and constructing a set of rules to control the search. Second, it provides a natural framework for planning the operations and monitoring the execution of those operations. In the case of failures, the failure points are easily identified and the system can return to the planner to select an alternative plan for retrieving the data.

1.2 Overview of SIMS

SIMS addresses several problems that arise when one tries to provide a user familiar only with the general domain with access to a system composed of numerous separate data- and knowledge-bases.

Specifically, SIMS deals with the following:

- Determining which information sources contain the data relevant to the knowledge-base classes used in formulating a given query.
- For those classes mentioned in the query which appear to have no matching information source, determining if any knowledge encoded in the domain model (such as relationship to other classes) permits reformulation in a way that will enable suitable information sources to be identified.
- Creating a plan, a sequence of subqueries and other forms of data-manipulation that when executed will yield the desired information.
- Using knowledge about databases to optimize the plan.
- In general, providing a uniform way to describe information sources to the system, so that data in them is accessible.

A visual representation of the components of SIMS is provided in Figure 1.2.[3]

An initial Loom query of the kind SIMS handles is shown in Figure 1.3. The first clause, `(rail_port ?port)`, is a concept expression that constrains the variable `?port` to a set of port objects in the knowledge base. The Loom class `rail_port` (standing for sea ports with rail facilities) need not necessarily correspond to the contents of a specific field in some single information source. If it does not, the planner will have to find some combination of subqueries that will obtain all necessary objects. This case is discussed further later. The second clause is a relation expression that states that the `port.refrig` relation holds between fillers of the variables `?refrig` and `?port`. This clause will bring about the retrieval of possible fillers of `?refrig` — refrigeration facilities in a relevant port. The third clause is a constraint: a ">" relation on the number of refrigeration facilities, requiring it to be a positive integer. The entire query requests the names of all ports with rail facilities and refrigeration facilities whose geographic code designation indicates that they are in Germany.

A fragment of the model describing some of the hierarchy of concepts relevant to this query is presented in Figure 2.2. In this figure, the circles denote concepts in the knowledge base, the upward arrows indicate *is-a* links, and the other arrows indicate relations between concepts. So, for example, the Port concept has two subconcepts, Sea_Port and Air_Port, and Sea_Port has a subconcept Rail_Port, seaports with a railway capability. Shaded concepts represent those that can be retrieved directly from some database.

If the information about rail ports and geographic locations were stored directly in the Loom knowledge base, then Loom could be used directly to answer this query. But, as the figure indicates, that is note the

[3] Work on the links back from the Execution component to Information Source Selection and Access Planning will not be discussed in this chapter.

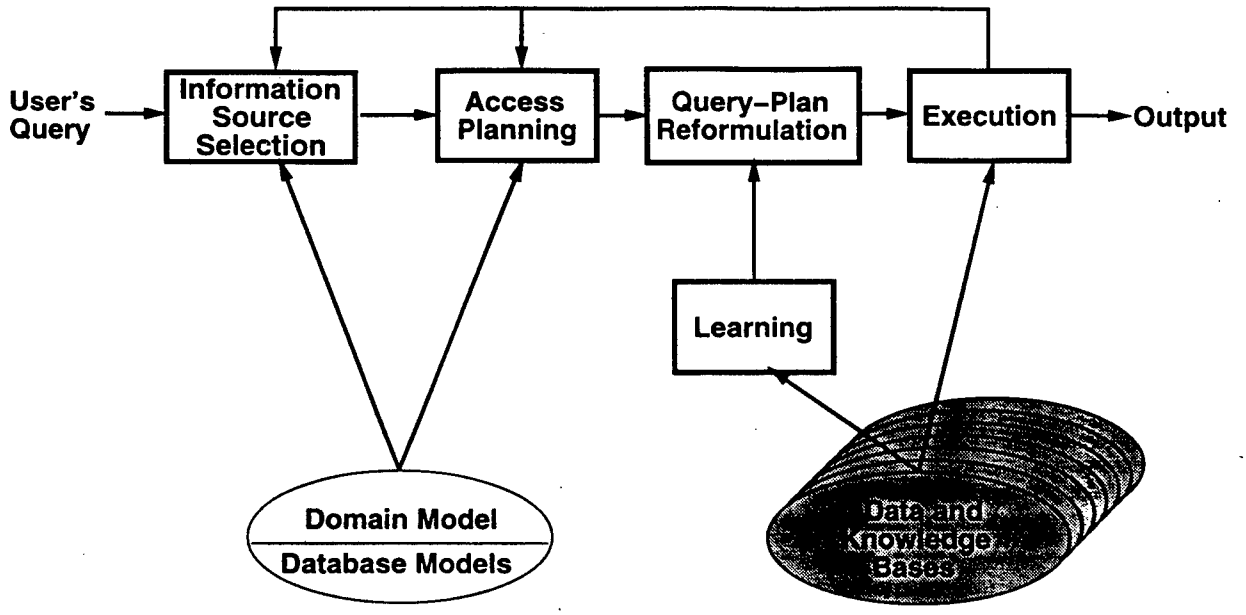


Figure 1.2: SIMS Overview Diagram.

```
(retrieve (?name)
     (:and (rail_port ?port)
           (port.refrig ?port ?refrig)
           (> ?refrig 0)
           (port.geocode ?port ?geocode)
           (port.name ?port ?name)
           (geoloc ?geoloc)
           (geoloc.country_name ?geoloc "Germany")
           (geoloc.geocode ?geoloc ?geocode)))
```

Figure 1.3: Example SIMS Query.

case. SIMS uses Loom to semantically model a domain about which data is stored in multiple information sources, and the information required to answer this query will be retrieved from the appropriate sources, with the help of LIM where necessary. Thus, if all the referenced information were stored in one database, this query could be passed directly to LIM as is. But that is not the case either.

Data pertaining to this query is spread over two databases — one containing information about ports and the other containing information about geographic locations. The system is handed the query shown in Figure 1.3 and it must first determine which information sources to access. Then it formulates a set of subqueries that can be executed directly by either LIM or Loom to derive the desired result. SIMS can use LIM to return intermediate results, which can then be processed further in Loom. As we will see, the execution of the example query will require three subqueries. One to each of the databases and one to combine the intermediate results obtained from them. The processes described in overview here are discussed in more depth in the remaining sections of the chapter.

The very first step in processing a query is to determine where the requested data resides. For instance, inspecting the model fragment in Figure 2.2 reveals that `rail_port` does not have a directly corresponding database (a shaded concept). However, the model relation `port.rail` can be used to distinguish it from other ports. Specifically, it can identify the desired ports from among those in `sea_port`, which *does* have a corresponding database. This and other reformulations of this nature are described further in Section 3.

The next step in processing the query is to produce a *plan* to implement the required retrieval. By this we mean that SIMS must produce a plan consisting of data-retrieval and data-manipulation specifications, with an associated partial ordering of the specified actions. The data-retrieval steps of the plan must be grounded

in specific information sources, i.e., all data one step requests must be contained in a single information source. Any data-manipulation steps of the plan are performed using the Loom reasoning facilities. The plan produced takes the form of a lattice of plan steps.

The steps in a plan are partially ordered based on the structure of the query. This ordering is determined by the fact that some steps make use of data that is obtained by other steps, and thus must logically be considered after them. For example, a plan step may compare two items of data according to some measure. If the data are obtained from two different information sources, then the comparison must come later than the retrievals of the data items.

Next, the plan produced as above is inspected and, when appropriate, data-retrieval steps that are grounded in the same information source are grouped — eventually their execution will result in a single query. We therefore call this process ***subquery formation***. The result of this grouping process is a new graph in which each node ultimately corresponds either to a query to some information source, or to internal manipulation by SIMS of data so acquired. The processes involved in subquery formation is described in Section 4.

After a plan for the query has been obtained, the system reformulates the query plan into a less expensive yet semantically equivalent plan. The reformulation is based on logical inference from content knowledge about each of the queried databases. The cost reduction from the reformulated plan is due to the reduction in the amount of the intermediate data and the refinement of each subquery. This *reformulation* process is described in Section 5.

Chapter 2

Retrieving and Integrating Data from Multiple Information Sources

1 Introduction

This chapter provides a general description of SIMS, in greater detail. We begin with a discussion of our approach to modeling. follow with descriptions of SIMS' planning and reformulation components, and conclude with a description of a demonstration interface to SIMS.

2 Domain and Information Source Models

SIMS must reason about data and other knowledge stored in a variety of locations and formats. It is imperative that SIMS have available detailed descriptions of the various information sources to which it has access. This is not merely an artifact of the SIMS approach — no system can retrieve requested information if it does not have knowledge about where the information in question may be stored and how to go about accessing it.

In SIMS a *model* of each information sources is created to describe it to the system. In addition, a *domain model* is constructed to describe objects and actions that are of significance in the performance of tasks in the application domain.[1] The domain model's collection of terms forms the "vocabulary" used to characterize the contents of an information sources.

It is important to note that the models of different information sources are independent of each other. This greatly simplifies the task of modeling, and at the same time enables new components to be added to SIMS without the need for any recompilation process. The planner simply makes use of the new information as appropriate.

2.1 Modeling Information Sources

For each information source, SIMS' model must include every fact that can influence decisions concerning when and whether to utilize it.

- In order to decide whether a query to LIM is necessary or whether processing can be performed locally, the model specifies if the source is a database or a Loom knowledge base (the two types of information sources currently supported);
- In order to decide whether to expend effort reformulating plans and whether to be concerned with the cost of transmitting intermediate data, the model describes the size of databases and tables, and their location;

[1] In fact, all the knowledge described here is stored by SIMS in a single model defined in a uniform way. It is thus only for purposes of exposition that we describe different parts of the model as "separate" models.

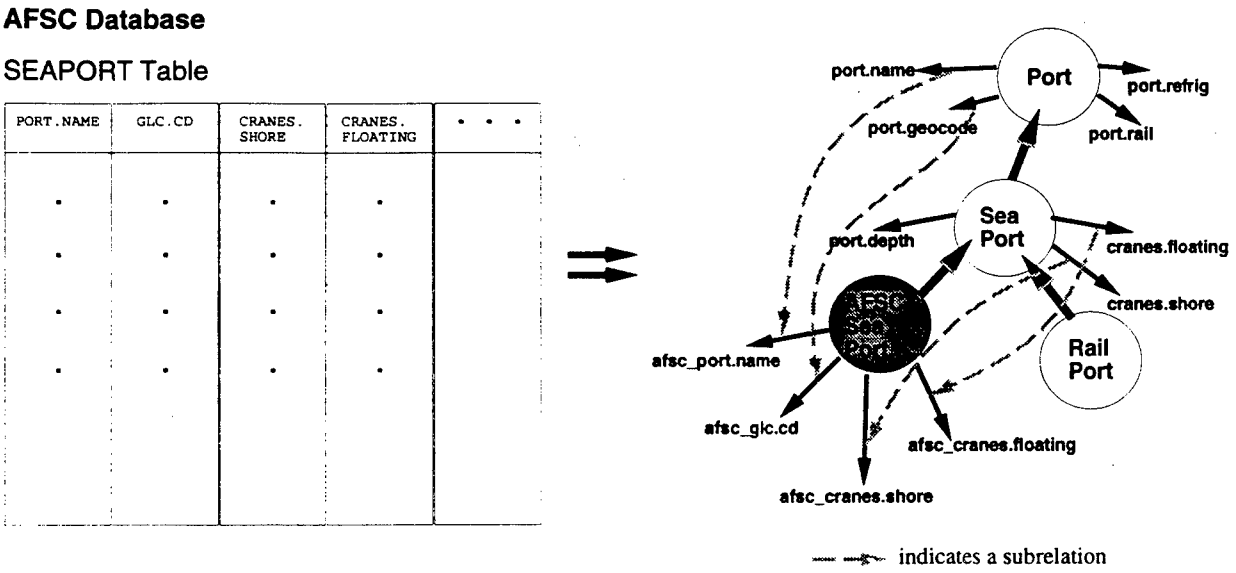


Figure 2.1: A Model of a Database Table Embedded in the Domain Model.

- In order to help further with decisions concerning reformulation, the model defines key columns in the database, if such exist; and, finally,
- In order to enable SIMS to determine in which information source desired information resides. the model describes the *content* of the information source.

In fact, most of the modeling effort done for SIMS goes to describing the content of databases. These models are used by both LIM and SIMS, for their own respective purposes (cf. [33] for LIM's work on database modeling). Simply put, the model of a database must describe precisely what type of information is stored in it. To do so we choose a key column (or columns) in each table and create a Loom class corresponding to it — the class from which items in that column are drawn. Every other column in the table is viewed as corresponding to a Loom relation — one describing the relationship between the key item and the one in that column. Figures 2.1 provides a simple illustration of content modeling.

2.2 Modeling the Domain

SIMS deals with a single "application domain", i.e., with organizing the retrieval of information relevant to some coherent collection of tasks. Currently, the application domain we have selected is the *military transportation planning* domain — tasks involving the movement of personnel and materiel from one location to another using aircraft, ships, trucks, etc.

SIMS' model of the application domain includes a hierarchical terminological knowledge base with nodes representing all objects, actions, and states possible in the domain. In addition, it includes indications of all relationships possible between nodes in the model. For example, there is a node in the model representing the class of ports and a node representing the class of geographic location codes. There is a relation specified between `ports` and `geoloc codes` with a notation indicating that each of the former has precisely one of the latter.

The Loom knowledge representation language is used to describe SIMS' domain model. Statements in Loom are used to express more elaborate relationships among model entities, such as that rail-ports are sea-ports which have a rail terminal as well (cf. Figure 2.2).[2]

[2] We have chosen simple examples for use in this chapter. Loom supports far more powerful statements. For a full description see [30, 31].

The entities included in the domain model are not meant to correspond to any objects described in any particular database. The domain model is intended to be a description of the application domain from the point of view of someone who needs to perform real-world tasks in that domain and/or to obtain information about it. However, the domain model is used, effectively, as the *language* with which to describe the contents of a database to SIMS. This is done by including relations — hierarchical (*is-a*) or others — to precisely describe every aspect of the contents of the database in terms of the domain model (cf. Section 2.1). In order to submit a query to SIMS, the user composes a Loom statement, using terms and relations in the domain model to describe the precise class of objects that are of interest. If the user happens to be familiar with particular databases and their representation, those concepts and relations may be used as well. But such knowledge is not required. SIMS is designed *precisely* to allow users to query it without such specific knowledge of the data's structure and distribution.

The task of accurately relating a database (and other information source) model must be engaged in for every database and knowledge base that SIMS is to be capable of utilizing. SIMS includes a graphical interface that simplifies this process (Section 6).

The modeling work that is a prerequisite for SIMS to be able to access information sources is a substantial effort, the importance of which cannot be over-emphasized. The extent to which SIMS can find information and the accuracy of its retrievals are completely dependent on it. The scalability of the modeling process in SIMS is discussed next.

2.3 Scalability and Expandability

SIMS' dependence on models of the domain and the information sources it utilizes requires that the question of its scalability be addressed. Separate issues arise when considering the application domain model and the information resource models.

Expanding the Application Domain Model

A considerable effort must be expended to model the application domain before any use of SIMS is possible. Although this task's extent should not be minimized, it is a relatively tractable one no different than that engaged in in many other areas of artificial intelligence. In fact, it has more clearly defined limits, since full utility is possible from the moment that enough of the model has been built to cover data objects described in desired databases. Any model building beyond that point only increases the expressivity of the query language and adds to the user's convenience, but it still provides access to the same data.

It is reasonable to anticipate that the domain model will have to be incrementally enlarged to accommodate new data sources as they are added to the system. However, since SIMS is designed to handle one domain at a time, it can safely be assumed that this modeling effort will gradually reach closure.

Adding Information Source Models

Additional modeling will have to be engaged in for every new information source added to SIMS. While this need will remain constant as the system grows, the SIMS approach greatly limits the required effort compared to what it potentially might be. Obviously, no approach to this problem can avoid modeling information sources, since without a complete description of the content of a database or knowledge base it is simply impossible to intelligently decide whether or not to attempt to retrieve desired information from it. However, SIMS allows one to model a new information source independently of any that are already incorporated into the system. There is no need to try to anticipate interactions or overlaps between different information sources, to decide how joins over databases will be performed, etc., since all such decisions are made at run time by the SIMS planner.

To further simplify any modeling that does have to be performed, the SIMS project includes an ongoing effort to develop modeling aids, among them a graphical Loom knowledge base builder (see Section 6).

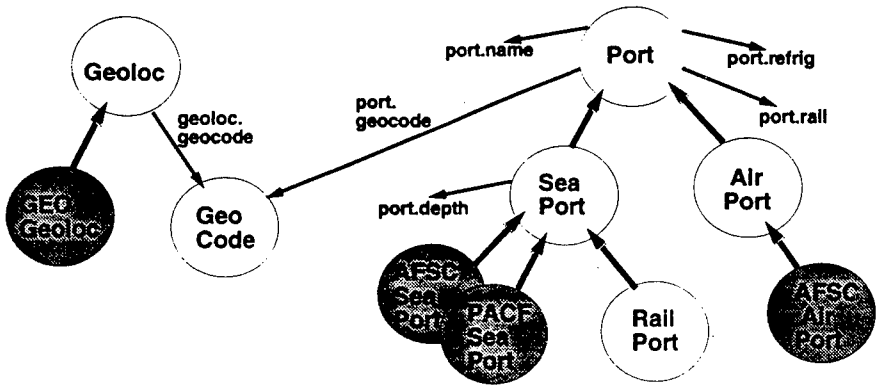


Figure 2.2: Fragment of Domain Model

3 Selecting Information Sources

The first step in answering a query expressed in the terms of the domain model is to select the appropriate information sources. This is done by mapping from the concepts in the domain model to the concepts in the database models that correspond directly to database information. If the user requests information about ports and there is a database concept that contains ports, then the mapping is straightforward. However, in many cases there will not be a direct mapping. Instead, the original domain-model query must be reformulated in terms of concepts that correspond to database concepts.

Consider the fragment of the knowledge base shown in Figure 2.2, which covers the knowledge relevant to the example query in Figure 1.3. The concepts Sea_Port, Air_Port, and Geoloc have subconcepts shown in by the shaded circles that correspond to concepts whose instances can be retrieved directly from some database. Thus, the AFSC database contains information about both seaports and airports and the PACF database contains information about only seaports. Thus, if the user asks for seaports, then it must be translated into one of the database concepts — AFSC_Sea_Port or PACF_Sea_Port. If the user asks for rail-ports, then it must first be translated into a request for sea_ports by augmenting the original query with a constraint that each port must have a railroad capability.

In addition to retrieving data from the databases, data can also be stored in and retrieved from the Loom knowledge base. This knowledge base is simply treated as another information source. However, the Loom KB has the added advantage that information from database queries can be cached in it and the model can be updated to indicate what information has been stored in Loom.

In this section, we describe the set of problem reformulation operations[3] that are implemented in SIMS and the reformulation process used to transform a user's query into one that can be used to retrieve data. We also describe how this reformulation mechanism supports the catching and retrieval of data in Loom.

3.1 Reformulation Operations

In order to select the information sources for answering a query, SIMS applies a set of reformulation operators to transform the domain-level concepts into concepts that can be retrieved directly from databases. The system uses four operators: Select-Database, Generalize-Concept, Specialize-Concept, and Partition-Concept. These reformulation operators are described next.

Select Database

The Select-Database reformulation operator maps a domain-level concept directly to a database-level concept. In many cases this will simply be a direct mapping from a concept such as Sea_Port to a concept that corresponds to the seaports in a particular database. There may be multiple databases that contain the

[3] These are to be distinguished from query-plan reformulation operations, which are described in Section 5.

same information, in which case the domain-level concept can be reformulated into any one of the database concepts. In Figure 2.2, Sea_Port can be transformed into either AFSC_Sea_Port or PACF_Sea_Port. The following example shows how a simple query would be reformulated using AFSC_Sea_Port. In general, the choice is made so as to minimize the number of queries to different databases.[4]

Input Query:

```
(retrieve (?name)
     (:and (sea_port ?port)
           (port.name ?port ?name)))
```

Reformulated Query

```
(retrieve (?name)
     (:and (afsc_sea_port ?port)
           (afsc_port.name ?port ?name)))
```

Generalize Concept

The Generalize-Concept operator uses knowledge about the relationship between a class and a superclass to reformulate a requested concept in terms of a more general concept. In order to preserve the semantics of the original request, one or more additional constraints may need to be added to the query in order to avoid retrieving extraneous data. For example, a request for rail ports can be replaced with a request for seaports with the additional constraint that the seaports have a rail capability (i.e. (port.rail ?port "Y")). This is illustrated in the following example.

Input Query:

```
(retrieve (?name)
     (:and (rail_port ?port)
           (port.name ?port ?name)))
```

Reformulated Query

```
(retrieve (?name)
     (:and (sea_port ?port)
           (port.name ?port ?name)
           (port.rail ?port "Y")))
```

Specialize Concept

The Specialize-Concept reformulation operator attempts to replace a given concept with a more specific concept. This is done by checking the constraints on the query to see if there is an appropriate specialization of the requested concept that would satisfy it. Identifying a specialization of a concept is implemented by building a set of Loom expressions representing each concept and then using the Loom classifier to find any specializations of the concept expression.

For example, consider the hierarchy fragment shown in Figure 2.2 again. Given the query shown below, which requests the ports with a depth greater than 25, the Loom classifier uses the fact that only seaports have a relation that corresponds to port.depth. Therefore, only seaports could possible satisfy the query, and in the original request ports can be replaced with seaports. There are several databases that correspond to seaports, so the requested information can now be retrieved.

Input Query:

```
(retrieve (?name)
```

[4] Currently we assume the databases contain consistent information, so the choice of databases only effects the efficiency of the query and not the accuracy.

```
    (:and (port ?port)
          (port.name ?port ?name)
          (port.depth ?port ?depth)
          (> ?depth 25)))
```

Reformulated Query

```
(retrieve (?name)
    (:and (sea_port ?port)
          (port.name ?port ?name)
          (port.depth ?port ?depth)
          (> ?depth 25)))
```

Partition Concept

The Partition-Concept operator uses knowledge about set coverings (a set of concepts that include all of the instances of another concept) to specialize a concept. This information is used to replace a requested concept with a set of concepts that cover it. For example, given the knowledge that the Port is covered by Sea_Port and Air_Port, a request for ports can be satisfied by retrieving and combining these two subconcepts. This is illustrated in the example below.

Input Query:

```
(retrieve (?name)
    (:and (port ?port)
          (port.name ?port ?name)))
```

Reformulated Query

```
(retrieve (?name)
    (:or (:and (sea_port ?port)
               (port.name ?port ?name))
         (:and (air_port ?port)
               (port.name ?port ?name))))
```

3.2 The Reformulation Process

Reformulation is performed by treating the reformulation operators as a set of planning operators and then using a planning system to search for a reformulation of the given set of concepts. The initial clauses of the query are divided into references to individual concepts and their associated constraints. The planner then searches for a way to map each of these concepts with their associated constraints into database concepts.

For example, consider the query shown below. It is first decomposed into two separate expressions – one about ports and the other about geolocs. Then the reformulation operators are used to find mappings to database concepts. Any remaining clauses (e.g., comparisons across concepts) are dealt with when a plan for accessing the data is generated.

```
(retrieve (?name)
     (:and (rail_port ?port)
           (port.refrig ?port ?refrig)
           (> ?refrig 0)
           (port.geocode ?port ?geocode)
           (port.name ?port ?name)
           (geoloc ?geoloc)
           (geoloc.country_name ?geoloc "Germany")
           (geoloc.geocode ?geoloc ?geocode)))
```

Using the reformulation operators described previously, the planner determines that the Geoloc concept expression can be mapped directly to a database and the Rail_Port concept expression needs to be reformulated. It can be reformulated into a Sea_Port concept expression, as described in Section 3.1, by adding a constraint. The resulting plan for reformulating the initial query is shown below.

```
generalize-concept rail_port (:and sea_port (filled-by port.rail "Y"))
select-database sea_port afsc_sea_port
select-database geoloc geo_geoloc
```

The final step is to take this plan and execute it. This is a straightforward process of applying the transformations in the query plan in the order listed. The resulting query is as follows.

```
(retrieve (?name)
      (:and (afsc_sea_port ?port)
            (afsc_port.rail ?port "Y")
            (afsc_port.refrig ?port ?refrig)
            (> ?refrig 0)
            (afsc_port.geocode ?port ?geocode)
            (afsc_port.name ?port ?name)
            (geo_geoloc ?geoloc)
            (geo_geoloc.country_name ?geoloc "Germany")
            (geo_geoloc.geocode ?geoloc ?geocode)))
```

3.3 Caching Retrieved Data

Data that is required frequently or is very expensive to retrieve can be cached in the Loom knowledge base and retrieved directly from Loom. An elegant feature of using Loom to model the domain is that caching the data fits nicely into this framework. The data is currently brought into Loom to perform the local processing, so caching is simply a matter of retaining the data and recording what data has been retrieved.

To cache retrieved data into Loom requires formulating a description of the data. This can be extracted from the initial query since queries are expressed in Loom in the first place. The description defines a new subconcept and it is placed in the appropriate place in the concept hierarchy. The data then become instances of this concept and can be accessed by retrieving all the instances of it.

Once the data is stored, it can be retrieved using the specialization operator that was described above. When the user poses the same query, the system can reformulate that query into the newly stored one and when the stored query is used, the cached data is retrieved directly from Loom.

4 Access Planning

The planning process described in this section finds an ordering of the database accesses and data comparisons by analyzing the dependency structure of the constraints on the query. It then generalizes the plan to remove any unnecessary ordering constraints in order to maximize the plan's potential parallelism. The complete database access plan is converted back into a partially ordered set of grounded subqueries that can be handed to LIM or executed directly in Loom. The first subsection below describes how the initial access plan is generated, and the second subsection describes how the plan is converted into the appropriate subqueries.

4.1 Generating an Access Plan

Since some of the databases are quite large, there can be a significant difference in efficiency between different possible plans for a query. Therefore, we would like to find subqueries that can be implemented as efficiently as possible. To do this the planner must take into account the cost of accessing the different databases, the cost of retrieving intermediate results, and the cost of combining these intermediate results to produce the final results. In addition, since the databases are distributed over different machines or even different

```
(and (concept afsc_sea_port ?port)
     (relation port.refrig ?port ?refrig)
     (relation port.geocode ?port ?geocode)
     (relation port.name ?port ?name)))
     (comparison > ?refrig 0)
     (concept geoloc ?geoloc)
     (relation geoloc.country_name ?geoloc "Germany")
     (relation geoloc.geocode ?geoloc ?geocode))
```

Figure 2.3: Goal Statement for the Planner

Operator	Purpose
`Retrieve-Concept`	Retrieves information from a particular database. subconcepts.
`Generate-Values`	Uses a given relation to generate values for a given variable.
`Filter-Values`	Uses a given relation to filter values for a given variable.
`Compare-Values`	Performs a comparison between two sets of values.
`Begin-Query`	Indicates the beginning of a query to one of the databases.
`End-Query`	Indicates the end of a query.

Figure 2.4: Operators for Planning a Query

sites, we would like to take advantage of potential parallelism and generate subqueries that can be issued concurrently.

A central task of the planner is to determine the ordering of the various accesses to databases. In the course of executing this task it also selects the databases from which to extract information. The ordering is determined by analyzing which steps in the plan for the query are generating values for variables and which steps are filtering the possible values. If one step depends on information produced in another step, then they must be done in the correct order. The Prodigy system, described in Section 1.1 is used to form the subqueries and order them. The problem is cast as a set of Prodigy operators, where the original semantic query constitutes the *goal* that is to be achieved by the planner.

In a straightforward process, the reformulated example query described in the last section is mapped by Prodigy into the goal for the planner shown in Figure 2.3. (Note that the language being used is no longer Loom.) Each subclause of the query is annotated with additional information indicating whether it is a concept, relation, or comparison subclause.

The set of operators used by the planner is shown in Figure 2.4. The first operator, **retrieve-concept** simply maps a concept to the database used to retrieve the desired information. The next three operators, **generate-values**, **filter-values**, and **compare-values**, determine the constraints on the order of the accesses to the individual databases. The remaining operators, **begin-query** and **end-query**, delimit the operations performed on an individual database.

As an illustration, the **retrieve-concept** operator is shown in Figure 2.5. This operator specifies a set of preconditions that must be true in order to apply the operator. In this case the preconditions are that information about the concept is directly available from some database and that this database has been opened. If the database has not been opened for retrieval, then the planner would create the subgoal of doing so and insert a **begin-query** operation. The **retrieve-concept** operator has two effects. The first specifies that the information for this concept is now available, and the second specifies in which database the information is available.

The system generates a plan to achieve the goal in Figure 2.3 by selecting operators to achieve each of the goal conditions. If the preconditions of a selected operator do not hold, then the system must recursively

```
(retrieve-concept
   (params (<pred> <object> <db>))
   (preconds (and (database-concpt <pred> <db>)
                  (open-db <db>)))
   (effects ((add (concpt <pred> <object>))
             (add (available <object> <db>)))))
```

Figure 2.5: Operator for Retrieving a Concept from a Database

achieve each of the preconditions. Once the system has achieved all of the goal conditions, it will have a plan for retrieving the information to satisfy the initial query. The resulting plan specifies which databases are to be used to satisfy the query as well as any constraints on the order in which the information is retrieved.

Prodigy initially produces a totally ordered plan for retrieving information. This plan is then converted into a partially ordered set of plan steps free of unnecessary ordering constraints. Each of an operator's preconditions in the database access plan explicitly states the conditions on which that operator depends. We use the algorithm of Veloso [58] to convert the totally ordered plan into a partially ordered plan from the definitions of the operators. This algorithm is polynomial in the length of the plan. The resulting partially ordered plan is shown in Figure 9.

4.2 Subquery Formation

The second step in the query planning process is to formulate the actual subqueries which will be passed on to LIM and eventually translated into database queries. Since LIM takes care of such details, we do not need to worry about the access languages of the individual databases. their locations, etc. Instead, we only need to formulate Loom queries that refer to information in one database. LIM and the DBMSs for the individual databases are responsible for selecting the appropriate access paths and locally optimizing the query within that database (we discuss global optimization in the next section).

The subqueries are formed by grouping together steps of the original plan. This is a relatively straightforward process that is aided by the presence of **begin-query/end-query** steps in the plan graph. The grouping is done by combining nodes in the plan partial order, to produce a final partial order on the subqueries. The subqueries for the example problem are shown in Figure 2.7. It shows that to implement the original query, three operations are necessary. The first two are accesses to separate databases that can be done in parallel. The third operation is a comparison in Loom on the results from these two subqueries. This last step cannot begin until the other two are complete.

5 Query-Plan Reformulation

Constructing a plan for retrieving information is only part of the problem. An important consideration in mapping the initial query into a set of subqueries is the total time that it will take to execute all of the subqueries. One approach to reducing this cost is to search for reformulations of the query access plan that reduce it. Database management systems (DBMSs) often perform syntactic query reformulation [25]. We leave that task to the respective DBMS then, and focus instead on more global semantic query reformulation [12, 27]. The idea is to transform the query resulting from the planning process into a semantically equivalent one that can be executed more efficiently.

Consider the planned query illustrated in Figure 2.7. The final step in this query, comparing two geographic location codes **?geocode** and **?geocode2**, could be quite costly since the cost of comparison is proportional to the square of the potentially large number of intermediate data items. Moreover, the comparison is performed in Loom, which is not as highly optimized as state-of-the-art DBMSs. There are a variety of ways in which this query could be reformulated to reduce or eliminate the cost of this last step. For example, knowledge about the contents in the databases could be used to augment the earlier subqueries, so that less intermediate information would be generated. Or, knowledge about the domain could be used to transform a subquery into an equivalent one that can be more efficiently executed.

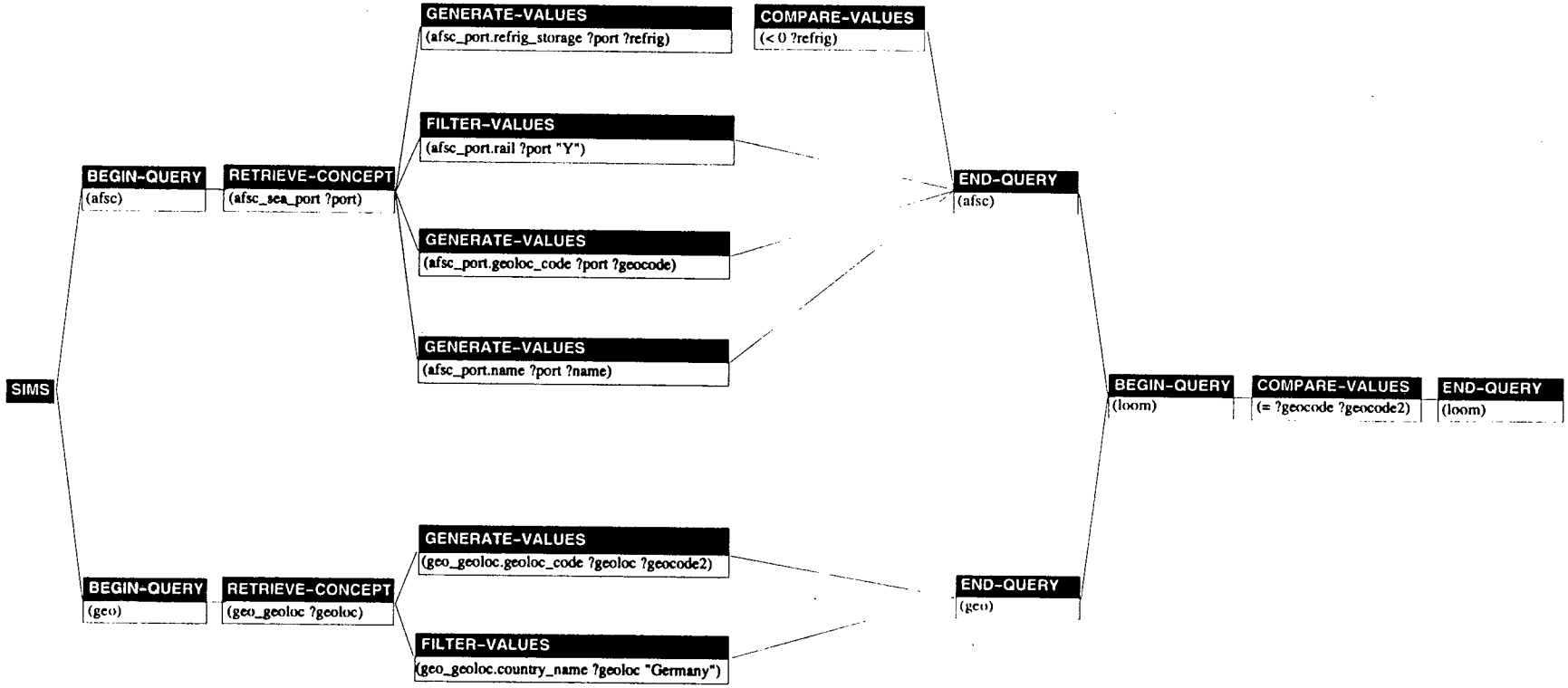


Figure 2.6: Preliminary SIMS Plan for Example Query

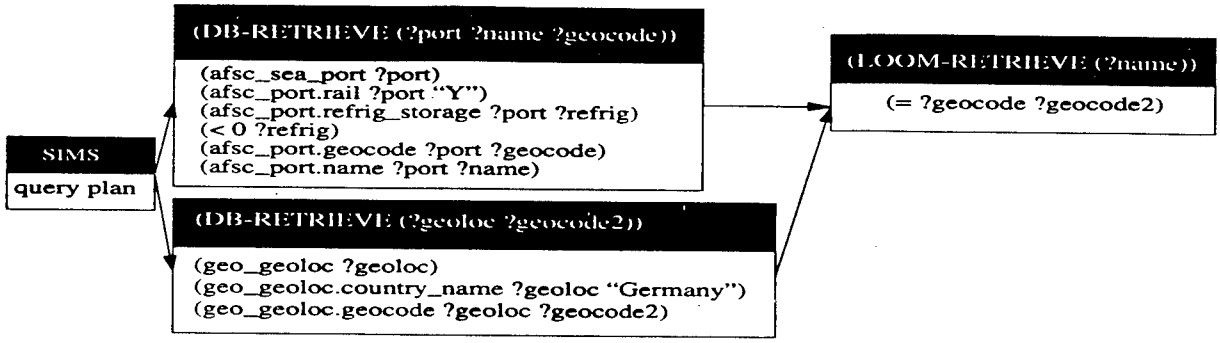


Figure 2.7: Final SIMS Plan for Example Query

Our approach to this problem differs from other related work on semantic query reformulation in an important respect that we do not rely on explicit heuristics of the database implementation to guide search for reformulations in the combinatorially large space of the potential reformulated subqueries. Instead, our algorithm considers all possible reformulations by firing all applicable rules and collecting candidate constraints in an *inferred set*. And then we select the most efficient set of the constraints from the inferred set to form the reformulated subqueries. This algorithm is not only more flexible and efficient, but the results of the rule firing turn out to be the useful information for extending subquery reformulation to the reformulation of the entire query plan. Most of other related work only reformulates single database queries.

Below we describe the principle behind the semantic reformulation, what knowledge is used for performing the reformulation, and the reformulation algorithms for subqueries and query plans.

5.1 Reformulation of Subqueries

The subquery reformulation problem is analogous to the problem of semantic query optimization for single database queries in previous work. The goal of query reformulation is to use reformulation to search for the least expensive query from the space of semantically equivalent queries to the original one. Two queries are defined to be ***semantically equivalent***[51] if they return identical answers given the same contents of the database. The alternative definition of semantic equivalence[27] requires that the queries return identical answers given **any** contents of the database, but this definition would limit us to using only semantic integrity constraints which are often not available. The use of the less restrictive definition of semantic integrity requires that the system updates the learned knowledge as the databases change.

The reformulation from one query to another is by logical inference using *database abstractions*, the abstracted knowledge of the contents of relevant databases. The database abstractions describe the databases in terms of the set of closed formulas of first-order logic. These formulas describe the database in the sense that they are true with regard to all instances in the database. We define two classes of formulas: *range information*, which are propositions that assert the value range of database attributes; and *rules*, which are implications with an arbitrary number of range propositions on the antecedent side and one range proposition on the consequent side. Figure 2.8 shows a small set of the database abstractions. In all formulas the variables are implicitly universally quantified.

The first two rules in Figure 2.8 state that for all instances, the value of its attribute country name is `"Germany"` if and only if the value of its attribute country code is `"FRG"`. With these two rules, we can reformulate the subquery `SUBQ1` in Figure 2.9 to the equivalent subquery `SUBQ2` by replacing the constraint on `geo_geoloc.country_name` with the constraint on `geo_geoloc.country_code`. We can inversely reformulate `SUBQ2` to `SUBQ1` with the same rules. Given a subquery Q, let $C_1, \ldots, C_k$ be the set of range and interaction constraints in Q, the following *reformulation operators* return a semantically equivalent query:

- **Range Refinement:** A range-information proposition states that the values of an attribute A are within some range R_d. If a range constraint of A in Q constrains the values of A in some range R_i,

Range Information:

```
1:(geo_geoloc.country_name∈("France" "Taiwan" "Japan" "Italy" "Germany"))
2:(afsc_port.geocode ∈ ("BSRL" "HNTS" "FGTW" "VXTY" "WPKZ" "XJCS"))
3:(0 ≤ afsc_port.refrig_storage ≤ 1000)
```

Rules:

```
1:(geo_geoloc.country_name = "Germany") ⟹ (geo_geoloc.country_code =
"FRG")
2:(geo_geoloc.country_code = "FRG") ⟹ (geo_geoloc.country_name =
"Germany")
3:(geo_geoloc.country_code = "FRG")⟹(47.15 ≤ geo_geoloc.latitude
≤ 54.74)
4:(afsc_port.rail = "Y" ) ⟹ (afsc_port.geocode ∈ ("BSRL" "HNTS"
"FGTW"))
5:(6.42 ≤ geo_geoloc.longitude ≤ 15.00) ∧
  (47.15 ≤ geo_geoloc.latitude ≤ 54.74)
  ⟹ (geo_geoloc.country_code = "FRG")
```

Figure 2.8: Example of Database Abstractions

SUBQ1:

```
(retrieve (?geoloc ?geocode2)
  (:and (geo_geoloc?geoloc)
        (geo_geoloc.geocode ?geoloc ?geocode2)
        (geo_geoloc.country_name ?geoloc "Germany")))
```

SUBQ2:

```
(retrieve (?geoloc ?geocode2)
  (:and (geo_geoloc?geoloc)
        (geo_geoloc.geocode ?geoloc ?geocode2)
        (geo_geoloc.country_code ?geoloc "FRG")))
```

SUBQ3:

```
(retrieve (?geoloc ?geocode2)
  (:and (geo_geoloc?geoloc)
        (geo_geoloc.geocode ?geoloc ?geocode2)
        (geo_geoloc.country_code ?geoloc "FRG")
        (geo_geoloc.latitude ?geoloc ?latitude)
        (?latitude >= 47.15) (?latitude <= 54.74)))
```

Figure 2.9: Equivalent Subqueries

then we can refine this range constraint by replacing the constraining range R_i with $R_i \cap R_d$.

- **Constraint Addition:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A, then we can add constraint B to Q.
- **Constraint Deletion:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A and B implies C_i, then we can delete C_i from Q.
- **Subquery Refutation:** Given a rule $A \rightarrow B$, if a subset of constraints in Q implies A, and in the query there exists a range constraint C_i such that B implies $\neg C_i$, then we can assert that Q will return `NIL`.

Replacing constraints is treated as a combination of addition and deletion. Note that these reformulation operators do not always lead to more efficient versions of the subquery. Knowledge about the access cost of attributes is required to guide the search. For example, suppose the only database index is placed on the attribute `geo_geoloc.country_name`. In that case reformulating SUBQ2 to SUBQ1 will reduce the cost from $O(n)$ to $O(k)$, where n is the size of the database and k is the amount of data retrieved. However, if either `geo_geoloc.country_name` and `geo_geoloc.country_code` are not indexed, then we will prefer

```
SUBQ-REFORMULATION(Subquery, DB-Knowledge, Cost-Model)
 1.refine range constraints, if Subquery refuted, return  Nil;
 2.for all applicable rules A  →  B in DB-Knowledge:
    if Subquery refuted, return  NIL;
    else add B to Inferred-Set, add (B,A) to Dependency-List;
 3.for all B in Inferred-Set in the order of their cost:
    if B is not indexed and ∃ (B,A) in Dependency-List
       delete B from Subquery, delete (B,A) from Dependency-List;
       replace all (C,B) in dependency list with (C,A);
 4.return (reformulated Subquery, Inferred-Set)
END.
```

Figure 2.10: Subquery Reformulation Algorithm

the lower cost short string attribute **geo_geoloc.country_code**. In this case, reformulating SUBQ1 to SUBQ2 becomes more reasonable. Figure 2.10 shows our subquery reformulation algorithm. We explain the algorithm below by showing how **SUBQ-REFORMULATION** reformulates the subquery SUBQ1, the lower query in the query plan in Figure 2.7.

There are three input arguments to the algorithm: the subquery to be reformulated, the database abstractions, and the cost model. The first step in the algorithm is to refine the range constraints. The only range constraint in SUBQ1 is on **geo_geoloc.country_name**, and its constrained value **Germany** is within the range of possible values (see the first formula of range information), so this constraint remains unchanged.

The second step is to match all applicable rules from the set of database abstractions using the reformulation operators defined above. The first rule in Figure 2.8 is matched and fired for SUBQ1 and we get an additional constraint (**geo_geoloc.country_code ?geoloc "FRG"**), which is added to the **Inferred-Set**. Then the second and third rules are matched because of the additional constraint on country code. The constraints **geo_geoloc.latitude** and **geo_geoloc.country_name** are added to the **Inferred-Set**.

The third step is to select the constraints in the **Inferred-Set** to delete from the subquery. The selection is based on the constraint's relative estimated execution cost which is computed by the type of the constraints (range constraint, or interaction constraint), the type of the attribute's values (integer, string, and their length), and whether they are indexed. The attribute **geo_geoloc.country_name** is deleted because its long string type is the most expensive. The next most expensive constraint is the one on attribute **geo_geoloc.country_code**. However, it should be preserved because the cause of its deletability (i.e., the constraint on **geo_geoloc.country_name**) was just deleted. Finally, the constraint on **geo_geoloc.latitude** is kept because it is an indexed attribute that will improve the efficiency of the subquery. The algorithm returns the reformulated subquery SUBQ3 as shown in Figure 2.9, as well as the **Inferred-Set**, which will be used for reformulating the succeeding subqueries in the query plan.

The worst case complexity of **SUBQ-REFORMULATION** is $O(R^2MN)$, where M is the maximum length of the antecedent of the rules, N is the greatest number of constraints in the partially reformulated query, that is, the number of original constraints plus the number of added constraints before final selection, R is the size of **DB-Knowledge**. In the average case, the complexity is much smaller than this worst case estimation. Because $R^2 \gg MN$, R is the dominating factor in the complexity and should be kept within a manageable size. This complexity analysis assumes that the system matches database abstractions by linear search. Therefore, a very large set of database abstractions could make the reformulation costly. To avoid this problem, we plan to adopt a more sophisticated rule match algorithm, such as the RETE algorithm[19], that will improve the algorithm's efficiency.

5.2 Reformulation of Query Plans

We can reformulate every subquery in the query plan with the subquery reformulation algorithm and improve their efficiency. However, the most expensive aspect of the multidatabase query is often processing intermediate data. In the example query plan in Figure 2.7, the constraint on the final subqueries involves the variables **?geocode** and **?geocode2** that are bound in the preceding subqueries. If we can reformulate these preceding subqueries so that they retrieve only those data instances possibly satisfying the constraint

```
QPLAN-REFORMULATION(Plan, DB-Knowledge, Cost-Model)
 1.KB ← DB-Knowledge;
 2.for all subqueries S in the order specified in Plan:
    (S',Inferred-Set) ← SUBQ-REFORMULATION(S,KB,Cost-Model);
    if S' refuted, return Nil;
    else update KB with Inferred-Set; update Plan with S';
 3.for all subqueries S whose semantics are changed:
    SUBQ-REFORMULATION(S, DB-Knowledge, Cost-Model);
 4.return reformulated Plan
END.
```

Figure 2.11: Query Plan Reformulation Algorithm

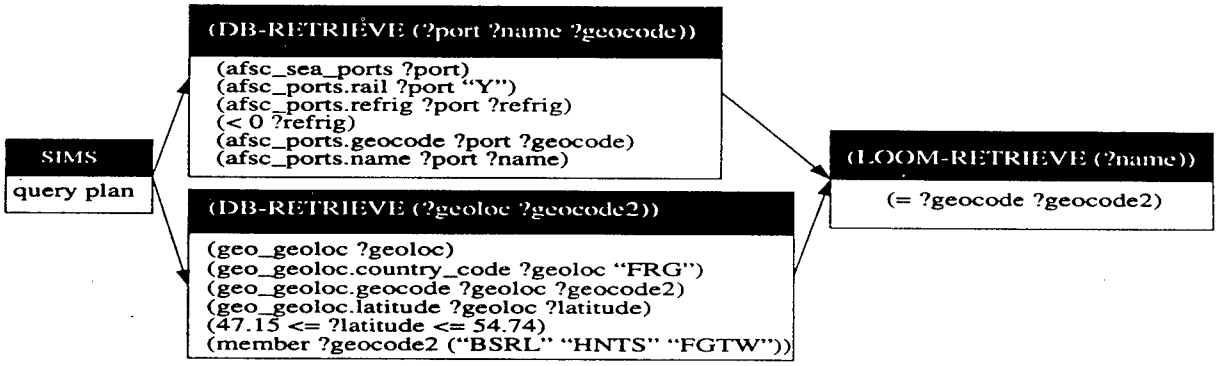


Figure 2.12: Reformulated SIMS Plan for Example Query

`(= ?geocode ?geocode2)` in the final subquery, the intermediate data will be reduced. This requires the query plan reformulation algorithm to be able to propagate the constraints along the data flow paths in the query plan. The query plan reformulation algorithm defined in Figure 2.11 achieves this by updating the database abstractions and rearranging constraints. We explain the algorithm below using the query plan in Figure 2.7.

The algorithm takes three input arguments: the query plan, the database abstractions, and the cost model. This algorithm reformulates each subquery in the partial order (i.e., the data flow order) specified in the plan using `SUBQ-REFORMULATION`. In addition, the database abstractions are updated with the `Inferred-Set` returned from `SUBQ-REFORMULATION` to propagate the constraints to later subqueries. In this example, the second formula of the initial range information is replaced by `(afsc_port.geocode` $\in$ `("BSRL" "HNTS" "FGTW"))`, the consequent condition of the fourth rule. The algorithm uses this updated range information to reformulate the final subquery and reduces the possible values from six to three. In addition, the constraint `(afsc_port.rail ?port "Y")` in the upper subquery is propagated along the data flow path to its succeeding subquery.

Now that the updated range information for `?geocode` is available, the subquery reformulation algorithm can infer from the constraint `(= ?geocode ?geocode2)` a new constraint `(member ?geocode2 ("BSRL" "HNTS" "FGTW"))`. In our example, the variable is bound by `(geo_geoloc.geocode ?geoloc ?geocode2)` in the lower subquery in Figure 2.7. The algorithm will insert the new constraint on `?geocode2` in that subquery. In this way, the constraints `(afsc_port.rail ?port "Y")` and `(= ?geocode ?geocode2)` are propagated back along the data flow path to the lower subquery. This process of new constraint insertion is referred to as *constraint rearrangement*. The final reformulated query plan is shown in Figure 2.12.

This query plan is more efficient than and returns the same answer as the original one. In our example, the lower subquery is more efficient because of the new constraint on the indexed attribute `geo_geoloc.latitude` (by `SUBQ-REFORMULATION`). The intermediate data items are reduced because of the new constraint on the attribute `geo_geoloc.geocode`. The logical rationale of this new constraint is derived from the constraints

query	1	2	3	4	5	6	7	8	9	10
planning time (sec)	0.5	0.3	0.6	2.1	1.1	0.7	0.7	0.5	0.5	0.8
reformulation time	0.1	0.1	0.0	0.5	0.1	0.0	0.0	0.1	0.1	0.3
rules fired (times)	37	18	11	126	63	8	17	15	19	71
query exec. time w/oR[a]	0.3	8.2	0.6	12.3	11.3	2.0	251.0	401.8	255.8	258.8
query exec. time w/R[b]	0.3	1.5	0.0	11.3	11.1	0.0	0.3	207.5	102.9	195.2
total elapsed time w/oR	0.8	8.5	1.2	14.4	12.4	2.7	251.7	402.3	256.3	259.6
total elapsed time w/R	0.9	1.9	0.6	13.9	12.3	0.7	1.0	208.1	103.5	196.3
intermediate data w/oR	-	-	-	145	41	1	810	956	808	810
intermediate data w/R	-	-	-	145	35	0	28	233	320	607

[a] w/oR = Without reformulation.

[b] w/R = With reformulation.

Table 2.1: Experimental Results

in the other two subqueries: `(afsc_port.rail ?port "Y")` and `(= ?geocode ?geocode2)`. and the fourth rule in the database abstractions.

The complexity of **QPLAN-REFORMULATION** is $O(SR^2MN)$, where S is the number of subqueries in the query plan, and R^2MN is the cost of **SUBQ-REFORMULATION**. In actual queries, S is relatively small, so the dominating factor is still the cost of the subquery reformulation R^2MN. in which the size of the database abstractions R is the most important factor, as shown in section 5.1. Thus, with a manageable size of the database abstractions, our algorithms are efficient enough to be neglected in the total cost of the multidatabase retrieval.

The earliest work in query reformulation was referred to as *semantic query optimization* and was applied to the single database query processing domain in a system called QUIST[27]. In contrast with *syntactic query optimization*, which has been widely studied in the database community, QUIST uses the rules of semantic integrity constraint of the database as background knowledge to reformulate the given query. However, QUIST and the following work[51, 12] use heuristics to select the reformulation operators and rules to reformulate the query in a hill-climbing manner. Our reformulation algorithm does not require heuristic control and is thus more flexible. Moreover, our algorithm utilizes the database abstractions to the greatest possible extent, while hill-climbing only searches for the local optimum.

5.3 Experimental Results of Reformulation

Table 2.1 provides statistical data concerning the preliminary experimental results of the query plan reformulation algorithm. In this experiment, the SIMS system is connected with two remote Oracle databases. One of the databases consists of 16 tables, 56,078 instances, the other database consists of 14 tables, 5,728 instances. The queries used were selected from the set of SQL queries constructed by the original users of the databases. The first three queries are single database queries, while the others are multidatabase queries. This initial results indicate that our algorithm can reduce the total cost of the retrieval substantially. In most multidatabase queries, the amount of intermediate data is reduced significantly. The overheads of reformulation is included in the total execution time and is relatively small compared to the cost of executing most queries.

The system used 267 database abstraction rules in this experiment. These rules were prepared by compiling the databases. The compiling procedure summarizes the range of each relation of the database by extracting the minimum and maximum values for numerical relations, or enumerating the possible values for string type relations. If the number of possible values exceeds a threshold, this range information is discarded. The rules were prepared by a semi-automatic learning algorithm similar to the KID3[43]. This algorithm takes a user input condition A, and learns a set of rules of the form $A \rightarrow B$ from the database. The algorithm retrieves the data that satisfy the condition A, then compiles the data for the conclusions B.

We are now developing an automatic learning algorithm that is driven by example queries. We plan to use inductive learning[10, 20, 34] to identify the costly aspects of the example subqueries, propose candidate rules to learn, and then refine the candidate rules to the desired operationality. Previous work that automatically derives the content knowledge is proposed by Siegel[51]. Our approach differs from theirs in that it is driven

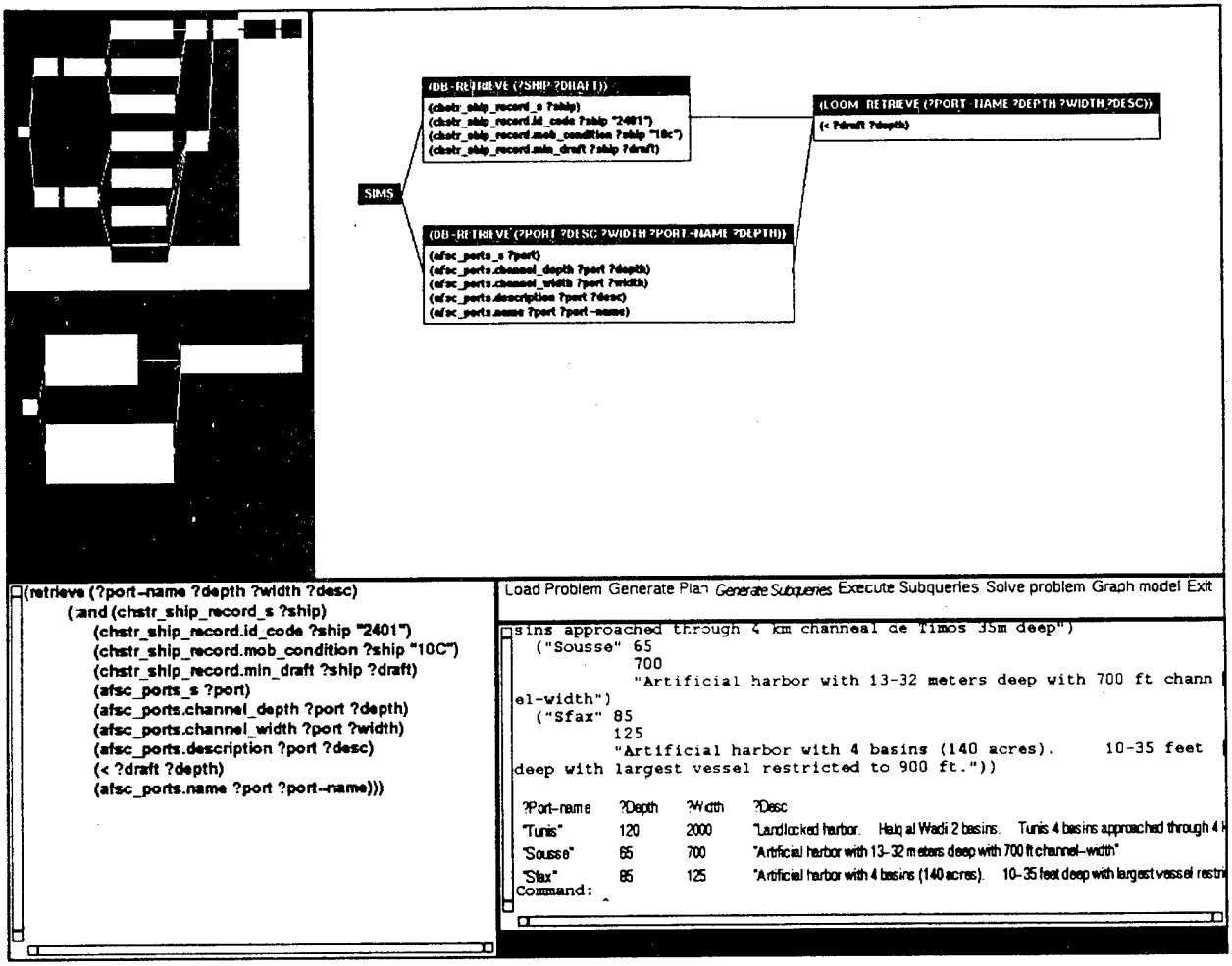


Figure 2.13: Sample SIMS Interface Screen

by the need for reformulation in the example instead of by a fixed set of heuristics, and it is flexible with regard to various database implementations. This is necessary in our case, since the databases integrated by SIMS are usually heterogeneous.

6 The SIMS Interface

Our intention is that the user view SIMS as a black box that allows the user to query multiple sources of information as if they were one single source. Given this scenario, an important issue is the ease with which the user can pose queries to the system and receive an answer. Since the exact terms used in a model by the developer will often differ from those which a user may be familiar with, it is very important that the developer's model be accessible to the user. We have thus stressed the importance of providing an easy to use interface for posing queries, one that allows the user convenient access to the model and help in construction of the query.

At the same time it is important that the model be constructed accurately by the developer. The model defines the application domain ontology, for both the user and SIMS. Not only will the model builder need to be able to build a good model of the domain, but he needs to be able to connect terms in the domain model with the corresponding terms in the database model. In order to build a model containing hundreds, possibly thousands of concepts, it is essential that the model builder have tools to view the models. The model builder also needs tools to help connect models fragments.

A common need for both users and model builders is a good way of viewing the model. Given that SIMS models are Loom models, a subsumption-based hierarchy of concepts, the logical visual representation to use is a graph. But a subsumption hierarchy only shows part of the definition of a model, the *is-a* relations, to show how a subconcept differs from its superconcept, it is usually necessary to show its roles. Hence our graph shows not only the concepts but the roles of concepts and their ranges.

SIMS does not dictate a single mode of interaction. We believe that the full range of underlying user interface management modalities should be made available to the user. Commands can be issued by mouse gestures applied to the desired objects, through a menu, or by keyboard commands. The user interface management system used by SIMS is CLIM 1.1, which is a high level presentation-based user interface system.

6.1 The Query Interface

SIMS is accessed by the user through the query interface. Central to the ability to pose a query is knowledge of the terms in which the domain is defined.

The SIMS query interface will provide the user aid in the following manner:

- A forms based query input facility.
- Access to the models via a graph of the domain and database models.
- The ability to specify terms of the query by clicking on nodes in the graph.
- Intelligent defaulting — automatic filling in of appropriate variable names for a sub-query.

6.2 The Model Building Interface

The domain model defines the ontology of the domain, i.e., all the terms and relations that one can use to query the various information sources. It also defines the expressiveness of the domain, as well as how powerfully SIMS can be in reformulating the queries. A domain model for a realistic application can easily contain hundreds of concepts and relations, and depending on the complexity of the application, can get out of hand very fast, especially if created using a text editor. At the same time, many concepts are likely to be very similar, being no more than slightly modified copies of already existing subconcepts of some concepts and hence tedious to enter. To ease the model building process, we provide the following tools:

- Two editors:
  - a form-based editor that is knowledgeable about the syntax of Loom terms and allowable inputs.
  - a text based editor for direct manual entry/modification of definitions.
- Interactive gesture-based editing, nodes can be modified, added or deleted by clicking the mouse on the relevant node.
- Graph navigation aids — panning, node hiding/unhiding and node centering.

7 Conclusions

This chapter describes a system for efficiently accessing and integrating information from multiple information sources (e.g., databases and knowledge bases). The various information sources are integrated using the Loom knowledge representation language. The system requires a model of the application domain and a model of the contents of each of the information sources. Then, given a query, the system generates and executes a plan for accessing the appropriate information sources. Before executing a query, the system first reformulates the individual subqueries to minimize the cost and the amount of intermediate data that is processed. Then the subqueries are executed, exploiting any parallelism in the plan.

SIMS currently integrates information from data stored in nine Oracle databases and information stored in a Loom knowledge base. The system uses the Loom Interface Manager (LIM) to retrieve data from the

Oracle databases and then processes all the data in Loom. The plan for selecting and accessing the various information sources is generated using the Prodigy planning system. The resulting plan is reformulated using a set of special purpose algorithms for semantic query optimization over multiple database queries.

Chapter 3

Generating Parallel Execution Plans with a Partial-Order Planner

1 Introduction

There are a wide variety of problems that require generating parallel execution plans. Partial-order planners have been widely viewed as an effective approach to generating such plans. However, strictly speaking, a partially-ordered plan represents a set of possible totally-ordered plans. Just because two actions are unordered relative to one another does not imply that they can be executed in parallel. The semantics of a partially-ordered plan provide that the two actions can be executed in either order. Simultaneous execution requires that potential resource conflicts between unordered actions be made explicit and avoided.

There are numerous partial-order planners presented in the literature, including SIPE [63], NONLIN [53], SNLP [32], UCPOP [42], TWEAK [14], O-PLAN [18], etc. Many of these planners have been used to produce parallel plans, but previously no one has precisely described the class of parallel plans they produce, identified their limitations, or described the assumptions made for the parallel plans that they do produce.

This chapter focuses on the use of partial-order planning to generate parallel execution plans. First, we identify the conditions under which two unordered actions can be executed in parallel. The component missing from many planners is an explicit representation of resources. Second, assuming that the resource constraints have been made explicit, we identify the classes of parallel execution plans that can be generated using different partial-order planners. Third, we present an implementation of a parallel execution planner based on UCPOP. Fourth, we describe how this planner can be used to generate parallel query access plans. Fifth, we compare the use of a partial-order planner to other approaches to building parallel execution plans. Finally, we review the contributions of the work and describe some directions for future research.

2 Executing Actions in Parallel

Classical planners assume that the execution of an action is indivisible and uninterruptible [62]. This is referred to as the atomic action assumption and stems from the fact that the STRIPS-style representation only models the preconditions and effects of an action. This assumption would appear to make simultaneous execution impossible, since it is unclear from the action model whether any two actions can be executed simultaneously without interacting with one another. This section identifies the conditions under which it is possible to execute two actions in parallel.

The work on parallelizing execution of machine instructions [54] provides some insight on the types of dependencies that arise between actions. Tjaden and Flynn identify three types of dependencies that must be considered in parallelizing machine instructions: procedural, operational, and data. A *procedural dependency* occurs when one instruction explicitly addresses another instruction and therefore imposes an ordering between the instructions. An *operational dependency* occurs when there is a resource associated with an instruction that is potentially unavailable. A *data dependency* occurs when one instruction produces

a result that is used by another instruction.

Similar dependencies arise in the parallelization of planning operations. A procedural dependency arises when one operation is explicitly ordered after another operation, which occurs in many of the hierarchical planners [53, 63] (e.g., see the `plot` construct in SIPE). This type of constraint is captured by explicit ordering constraints between actions. A data dependency arises when the precondition of one operation depends on the effects of another operation. This type of dependency is captured by the operator representation and corresponding algorithms, which ensure that if two actions are unordered relative to one another, their preconditions and effects are consistent. Operational dependencies can occur when there are limited resources associated with an operation. This type of dependency is often ignored by planning systems.

Executing actions in parallel requires explicit handling of potential resource conflicts. If two actions are left unordered in a partial-order plan, they can be executed in either order. In order to execute them in parallel, we must ensure that there are no potential conflicts that occur during execution. Most conflicts will be resolved in the process of ensuring that the preconditions and effects are consistent. However, because of the limited representation, the type of conflict that is not typically handled in a partial-order planner is when two actions require the same reusable resource. This type of resource conflict is not typically captured by the preconditions and effects because at the start of execution the resource is available and when execution completes it is available.

Despite the problem of potential resource conflicts, a number of partial-order planners have allowed simultaneous execution. They do so by either assuming the actions are independent [53], augmenting the action representation to avoid resource conflicts [63], or requiring the user to explicitly represent the conflicts in the preconditions and effects of the operators [18]. The approach of simply assuming that the actions are independent could lead to unexpected resource conflicts. The approach of requiring the user to represent the conflicts in the preconditions and effects is both awkward and computationally more expensive, since it requires additional operators that explicitly allocate and deallocate resources. The most natural approach is to augment the action representation to describe the explicit resource needs of the different actions. This approach was proposed in SIPE [63], where each operator can be annotated to explicitly state if something is a resource. In the next section we will assume that the resource constraints have been made explicit and in the following section we will describe our approach to representing and reasoning about resources.

3 Parallel Execution Plans

This section identifies the classes of parallel execution plans that can be generated by different planners, assuming that a domain is correctly axiomatized and explicitly represents the resource requirements of the operators. The different types of parallel execution plans can be broken down into several classes, ranging from plans with completely independent actions to those where the actions must be executed in parallel or must overlap in a particular manner in order for the plan to succeed. As the interactions in the plan increase in complexity, the corresponding planners require more sophisticated representations and reasoning in order to generate such plans. In this section we present four classes of parallel execution plans and identify the corresponding planners that can generate that class of plans. These classes are described in order from the most restrictive to the least restrictive.

3.1 Independent Actions

The most restricted type of parallel execution plans are those where all of the parallel actions are completely independent of one another.

> Two actions are defined to be ***independent*** if and only if the effects of the two actions executed simultaneously are the union of the individual effects of the actions done in isolation.

Allen [1] notes that various partial-order planners, such as NONLIN [53], DEVISER [60], and SIPE [63], all "allow simultaneous action when the two actions are completely independent of each other." While this statement is correct, it is a bit misleading since these planners can generate plans for a less restrictive class of parallel plans. As noted by Horz [21], since some of the effects of an operator may be unnecessary with respect to the goal and preconditions of other operators, the fact that two operators are unordered in a plan

generated by a partial-order planner does not imply that they are independent. A planner that can only generate plans with independent actions is UA [36], which imposes ordering constraints between any pair of unordered actions that could possibly interact.

Figure 3.1 illustrates a simple plan with two independent actions. The goal of the plan is to have the table painted red and the chair painted blue. Since the actions of painting the table and painting the chair are independent, they can be executed in parallel.

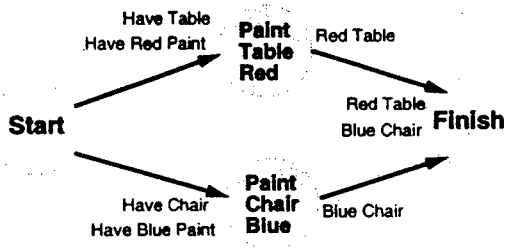


Figure 3.1: Plan With Independent Actions

3.2 Independent Actions Relative to a Goal

In a variety of partial-order planners, such as SIPE [63], SNLP [32], and UCPOP [42], the planners enforce the property that two actions can only remain unordered if there is no threat between them. A threat occurs when an operator could potentially delete a condition that is relevant to achieving the final goal.[1] A condition is defined to be *relevant* if and only if it is a goal condition or a precondition of an operator that in turn achieves a relevant condition. The class of parallel plans produced by these planners are those with *independent actions relative to the goal.*

> Two actions are *independent relative to a goal* G if and only if, for all conditions that are relevant to achieving G, the result of executing the actions in either order is identical to the result of executing the actions simultaneously.

These planners are limited to this class of plans since, if there is a threat between any pair of actions, additional constraints are imposed on the plan to eliminate the threat.

Figure 3.2 shows an example plan where all of the actions are independent relative to the goal. This example differs from the independent action example in that the two painting actions each have a side-effect of painting the floor as well as the object. Thus, the paint table and paint chair operators are not independent since both operations also paint the floor different colors. However, since the color of the floor is irrelevant to the goal of getting the table painted red and the chair painted blue, the plan is still valid and could be generated by planners in this class.

3.3 Independent Subplans Relative to a Goal

Not all partial-order planners enforce the property that two actions can remain unordered only if there are no threats between them. In particular, those planners that implement some form of Chapman's white knight [14] require only that there exist some operator that establishes a given precondition, but do not commit to which operator. More specifically, the white knight operation allows plans with the following conditions: There exists some operator op_1 that achieves a goal or precondition g. There exists a second operator op_2 that possibly deletes g. And there exists a third operator op_3 that follows op_2 and achieves g. If we are interested in producing totally-ordered plans, then the white knight operator is not required for

[1] Some planners, such as SNLP and earlier versions of UCPOP, defined a threat to include an operator that *adds* or deletes a relevant condition. This stronger definition of a threat is used to constrain the search space and would prevent some possible parallel plans from being generated.

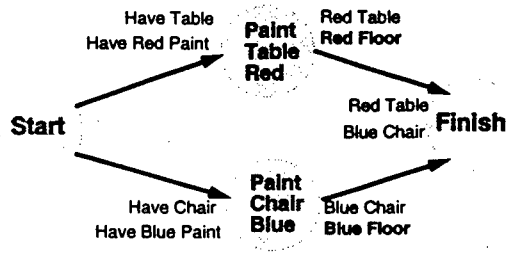


Figure 3.2: Plan with Independent Actions Relative to the Goal

completeness. However, the use of the white knight operator allows a planner to generate a slightly more general class of parallel plans.

The planners in this class include TWEAK [14], NONLIN [53], O-PLAN [18], MP, and MPI [26]. The class of parallel plans produced by these planners are those with *independent subplans relative to a goal.*

> Two *subplans* are *independent relative to a goal* G if and only if, for all conditions that are relevant to achieving G, the result of executing the *subplans* in either order is identical to the result of executing the *subplans* simultaneously.

The class of parallel plans that can be generated by the planners in this class, but cannot be generated by the planners in the previous class are those where there are actions that are not independent, but the subplans in which the actions occur are independent.

Figure 3.3 shows an example plan with independent subplans relative to the goal (adapted from an example in [26]). In this example, before the table and chair can be painted red, they must be primed, and priming them has a side effect of painting the floor white. The final goal of the problem is to get the table, chair, and floor all painted red. Notice that the action of priming the chair interacts with painting the table, since they both change the color of the floor. Similarly, priming the table interacts with painting the chair. Despite these potential interactions, the floor will still be painted red at the end of the plan since the table and chair must be painted after they are primed. Solving this problem requires the white knight operation to produce the parallel plan since the plan does not state which painting operation will be used to achieve the final goal of making the floor red.

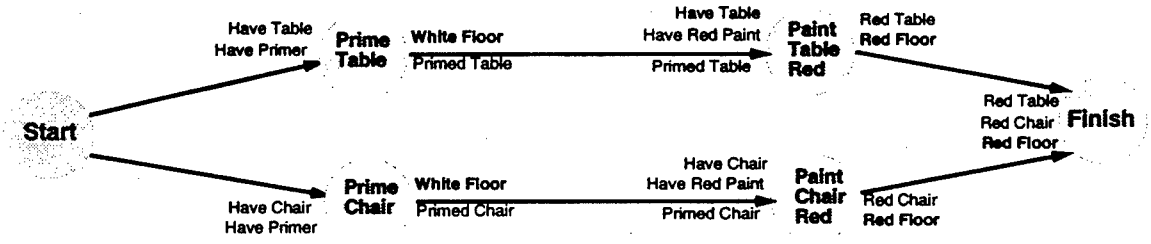


Figure 3.3: Plan with Independent Subplans Relative to the Goal

The implementation of the white knight, which allows a planner to generate this more general class of parallel plans, also makes it difficult to extend the operator language to efficiently handle more expressive constructs, such as conditional effects and universal quantification [14]. These more expressive language constructs are often required for representing and solving real problems.

3.4 Interacting Actions

The most general class of parallel plans are those where the parallel actions interact in some way. Two actions may need to be executed in parallel or two actions may need to overlap in a particular manner in order for the plan to succeed. For example, if the final goal was to get the chair blue, the table yellow, and

the floor green and there was no green paint, we could paint the table and chair simultaneously. To handle these cases requires the introduction of an explicit representation of time, such as that provided in temporal planning systems [1, 41]. However, in this chapter we are interested in the more restricted case where we would like to execute actions in parallel to take advantage of the possible parallelism to reduce the total execution time, not because the solution requires parallelism to solve the problem.

4 Parallel Execution Planning in UCPOP

We used the UCPOP planner [42, 7] to build a parallel execution planner. The analysis in the previous section showed that UCPOP can produce the class of plans with actions that are independent relative to a goal. For the specific application described in the next section, this restriction does not prevent the system from finding any solutions. The changes to UCPOP that were required were to add explicit resource definitions to the operators, to modify the planner to enforce the resource constraints, and to construct an evaluation functions to estimate the cost of the parallel plans.

The resource requirements of the operators are made explicit by augmenting each operator with a resource declaration. An example operator with a resource declaration is shown in Figure 3.4. This operator describes the action of moving data from one data source to another and declares the data source from which the data is being moved as a resource. The purpose of this declaration is to prevent one operator from being executed in parallel with another operator that requires the same database.

```
(define (operator move-data)
    :parameters (?db1 ?db2 ?data)
    :resources ((resource ?db1 database))
    :precondition (:and (available ?db1 ?data)
                        (:neq ?db1 ?db2))
    :effect (:and (:not (available ?db1 ?data))
                  (available ?db2 ?data)))
```

Figure 3.4: Operator with Resource Declaration

In order to avoid resource conflicts, we modified the planner to ensure that if two operators require the same resource, then they are not left unordered relative to one another. In SIPE this is done with a critic that checks for resource conflicts and then imposes ordering constraints when conflicts are found. In UCPOP, we added a check to the planner such that every time a new action is added to the plan, the planner checks for potential resource conflicts with any other operator that could be executed in parallel. Any conflicts discovered are added to the list of threats that must be removed before the plan is considered complete. Using the search control facility in UCPOP, these conflicts can be resolved immediately or delayed until later in the planning process.

Since efficiency is the primary motivation for generating parallel plans, we constructed an evaluation function that can be used to find plans with low overall execution time. Since this evaluation function underestimates the cost of the parallel plan, the planner can use a best-first search to find the optimal plan. This evaluation function takes into account that the cost of executing two actions in parallel will be the maximum and not the sum of the costs. The space of parallel execution plans may be quite large, so domain-specific control knowledge may be necessary to search this space efficiently.

The evaluation function to determine the execution time of a parallel execution plan is implemented using a depth-first search. The search starts at the goal node and recursively assigns a cost to each node in the plan. This cost represents the total cost of execution up to and including the action at the given node. The cost is calculated by adding the cost of the action at the node to the maximum cost of all the immediately prior nodes. Once the cost of the plan up to a node has been computed, we store this value so it will only need to be calculated once. Since each node (n) and each edge (e) in the graph is visited only once, the complexity of evaluating the plan cost is O(max(n,e)).

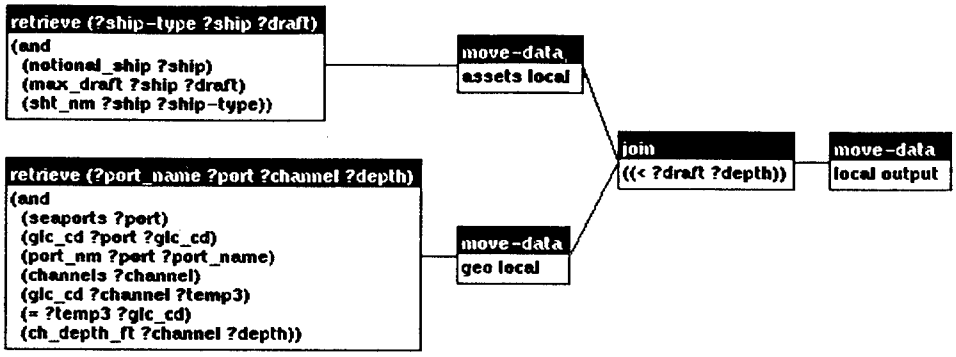


Figure 3.5: Parallel Query Access Plan

5 Parallel Query Access Plans

There are two general characteristics of a domain where the use of a partial-order parallel-execution planner will be useful and effective. First, it is applicable to those domains where the actions could be executed serially, but the overall execution time can be reduced by executing some of the actions in parallel. Second, it will be most useful in those domains where the choice of the operations determines or limits the overall execution time of the plan. As such, the plan generation and scheduling cannot be done independently since this would potentially result in highly suboptimal plans.

We applied the parallel execution planner to a query access planning problem that involves multiple distributed information sources [28]. In this domain, a plan is produced that specifies how to retrieve and generate the requested set of data. This requires selecting sources for the data, determining what operations need to be performed on the data, and deciding on the order in which to execute the operations. The planner must take into account the cost of accessing the different information sources, the cost of retrieving intermediate results, and the cost of combining these intermediate results to produce the final results. A partial-order parallel-execution planner is ideally suited for this problem since the parallelization is for efficiency purposes, and there are many possible plans for retrieving the same data and the choice of plans is crucial in determining the overall efficiency.

Figure 3.5 shows an example parallel query-access plan. The three basic query access planning operations used in this plan are *move-data*, *join*, and *retrieve*. The *move-data* operation moves a set of data from one information source to another. The *join* operation combines two sets of data into a combined set using the given join relations. The *retrieve* operation specifies the data that is to be retrieved from a particular information source.

The domain and planner described here are fully implemented and serve as an integral part of an information retrieval agent. We have also extended UCPOP to perform execution, and to do so in parallel. The system is implemented and runs in Lucid Common Lisp on SUN and HP workstations. To provide a sense for the potential speed-up of this approach we ran a sample query that involved queries to two different databases. Without parallelization, the system generated a plan with six operators in 0.82 CPU seconds and then executed the plan in 101.8 seconds of elapsed time. With parallelization, it generated the plan in 1.3 CPU seconds and executed the plan in 62.4 seconds of elapsed time, a 39 percent reduction in execution time.

6 Related Work

An alternative approach to addressing the problem of simultaneous execution is provided by work on temporal planning [1, 41]. A temporal planner can handle the general problem of simultaneous parallel execution, but this general solution has a cost, since just testing the satisfiability of a set of assertions is NP-hard [61]. The capabilities of a full-fledged temporal planner are necessary only if we need to explicitly reason

about the interaction between parallel actions. In this chapter we focus on the simpler problem of non-interacting simultaneous execution, which does not require a full-blown temporal reasoner to handle. In fact, partial-order planners appear to be well suited for problems in this class.

Another approach to this problem is to generate totally-ordered plans and then convert each plan into a partially-ordered plan [59, 45]. The problem with this approach is that the particular choice of the totally-ordered plan determines the parallel execution plan. As such, in order to consider the space of parallel execution plans requires searching through the space of totally-ordered plans. Since a single partially-ordered plan often corresponds to a number of totally-ordered plans, it will be harder to efficiently search the space of parallel execution plans.

Recently, Backstrom [6] showed that the general problem of finding an optimal parallel execution plan is NP-hard. We cannot escape from this complexity result; however, partial-order planners do avoid the NP-hard subproblem of testing satisfiability and provide a more natural framework than total-order planners for searching the space of parallel plans and encoding domain-specific control knowledge to guide the search.

7 Discussion

The idea of using partial-order planning to generate parallel execution plans has been around since the early days of planning. What we have done in this chapter is to explicate the underlying assumptions and situations where parallel execution is possible, characterize the differences in the plans produced by various planning algorithms, and identify the changes required to use UCPOP as a parallel execution planner. We have also shown that these ideas apply directly to the problem of generating parallel query access plans.

In future work we plan to tightly integrate the planning and execution components. This would allow the system to dynamically replan actions that fail, while continuing to execute other actions that are already in progress. In addition, we plan explore the problem of how to efficiently search the space of parallel execution plans. First, we will consider domain-independent search strategies that produce the highest quality solution that can be found within the time allotted. Second, we will exploit domain-specific knowledge to both restrict the search space and guide the search.

Chapter 4

Reformulating Query Plans For Multidatabase Systems

1 Introduction

An important and difficult problem is how to efficiently retrieve information from distributed, heterogeneous multidatabase systems (Sheth and Larson, 1990). Retrieving and integrating distributed data often requires processing and storage of large amounts of intermediate data, which can be very costly. This cost can be reduced in some cases by selecting the appropriate sites for processing and employing query optimization techniques (Apers, Hevner, and Yao, 1983; Jarke and Koch, 1984) to reduce the cost of individual queries. However, these techniques are often inadequate since they rely on limited information about the syntactic structure of the queries and databases. This information alone is not usually sufficient for reducing the cost for complicated distributed, heterogeneous multidatabase queries.

This chapter addresses this problem of multidatabase retrieval by bringing to bear a richer set of knowledge about databases to optimize multidatabase queries. The idea is to use semantic knowledge of the contents of databases to reformulate queries into equivalent yet less expensive ones. Using the additional semantic knowledge, the potential cost reduction is significantly greater than can be derived from optimization based on the syntactical structure of queries alone. Since the knowledge required can be learned from any database, this approach is very general.

Consider the following hypothetical example. Suppose that there are two databases in a multidatabase system, one containing data about ports, and another about ships. A query is given to retrieve the data of ports that have a depth that can accommodate tankers. This query may be very expensive because the data about ports must be retrieved and compared with the draft of all the tankers. Suppose that the system learned from the ship database that if the ship type is tanker, then its draft is at least 10 meters. With this knowledge, the original query can be reformulated so that the system only retrieves data about ports whose depth is greater than 10 meters. This additional constraint may significantly reduce the amount of data retrieved from the ship database and thus substantially reduce the cost of executing the query.

In this chapter, we present an efficient algorithm to perform this type of semantic reformulation. We implement the algorithm in the context of the SIMS project (Arens and Knoblock, 1992; Arens, Chee, Hsu, and Knoblock, 1993). The SIMS project applies a variety of AI techniques and systems to build an integrated intelligent interface between users and distributed, heterogeneous multiple data/knowledge-bases systems. Given a multidatabase query, the planner of SIMS generates a partially ordered query plan to retrieve the data. The reformulation algorithm presented here is used to reformulate this initial query plan to reduce the cost of retrieval.

The query reformulation approach was initially proposed by (King, 1981) and (Hammer and Zdonik, 1980). Our approach differs from theirs and the following related work (Siegel, 1988; Chakravarthy, Grant and Minker, 1990) in that we do not rely on heuristics to guide the search in a hill-climbing manner, which often results in local optima. Moreover, we consider queries for data distributed over multiple sources, while they only consider single database queries.

The remainder of this chapter is organized as follows. The next section describes the query planning in SIMS. Section 3 reviews the semantic query optimization and our reformulation algorithm for single database queries. Section 4 extends the idea to multidatabase queries. Section 5 shows our experimental results. We compare our approach with related work in Section 6. Section 7 reviews the contributions of the work and describes directions for future work.

2 Query Planning

Figure 4.1 shows an example SIMS semantic query. This query retrieves the name of the ports in Germany that have both railroad capabilities at the port and refrigerator storage. SIMS accepts queries in the form of a description of a class of objects about which information is desired. This description is composed of statements in the Loom knowledge representation language (Macgregor, 1990). The user is not presumed to know how information is distributed over the data- and knowledge bases to which SIMS has access — but he/she *is* assumed to be familiar with the application domain, and to use standard terminology to compose the Loom query. The interface enables the user to inspect the domain model as an aid to composing queries. SIMS proceeds to decompose the user's query into a collection of more elementary statements that refer to data stored in available information sources. SIMS then uses Prodigy (Carbonell, Knoblock, and Minton, 1991) to create a plan for retrieving the desired information, establishing the order and content of the various plan steps/subqueries. Figure 4.2 shows an example partially ordered multidatabase query plan generated by SIMS's query planner.

```
(retrieve (?name)
  (:and (port ?port)
        (port.rail ?port "Y")
        (port.refrig ?port ?refrig)
        (> ?refrig 0)
        (port.geocode ?port ?geocode)
        (port.name ?port ?name)
        (geoloc ?geoloc)
        (geoloc.geocode ?geoloc ?geocode)
        (geoloc.country_name ?geoloc
         "Germany")))
```

Figure 4.1: Example SIMS Semantic Query

Each node in the plan corresponds to a *subquery* to an individual data- or knowledge base. The edges indicate the data flow direction from one database to another. Data pertaining to this query is spread over two remote databases — one containing information about ports and the other about geographic locations. In the figure, the two **db-retrieve** subqueries are queries to each of these databases. They will be translated into their corresponding database query languages before being sent to the DBMSs. The **loom-retrieve** subquery contains the interaction constraints involving values from the different remote databases. To execute this query plan, the two **db-retrieve** subqueries will first be executed, and the retrieved data will be loaded into Loom and translated into objects of semantic classes. Loom then evaluates the constraints specified in the **loom-retrieve** to retrieve the resired answer from the sets of intermediate data.

Each subquery consists of conjunctions of constraints. In the upper subquery in Figure 4.2, the first clause, **(afsc_sea_port ?port)**, binds the variable **?port** to the set of port instances in the database **afsc_sea_port**. The second clause is a *range constraint* which restricts the attribute **afsc_port.rail** of **?port** to have value **Y**. This indicates that the port has railroad capability. The clause **(> ?refrig 0)** is another example of range constraint that constrains the ports to have non-zero refrigerator storage. Constraints that involves two or more variables are *interaction constraints*, such as the one in the **loom-retrieve** subquery.

The most expensive part of the query plan is often moving intermediate data from remote databases to the local Loom system. Consider the example in Figure 4.2, the cost of pairwise comparison in the final subquery is proportional to the square of the amount of data items retrieved from the remote databases.

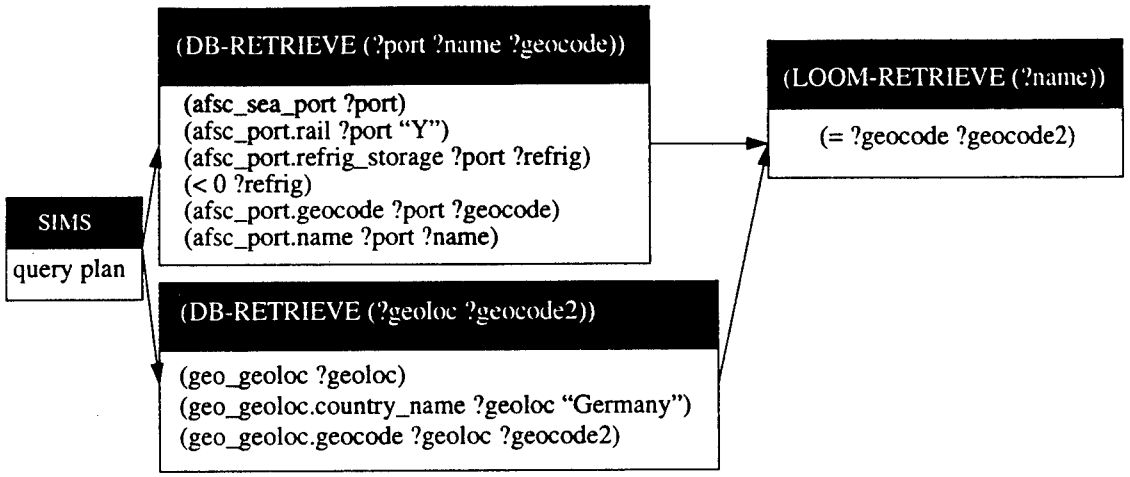


Figure 4.2: Preliminary SIMS Plan for Example Query

If we can reformulate these subqueries such that the interaction constraints in the final subquery are also considered, the amount of intermediate data will be reduced.

3 Subquery Reformulation

We start with the subquery reformulation algorithm and then extend it to reformulate the entire query plan in the next section. The goal of the query reformulation is to use reformulation to search for the least expensive query from the space of semantically equivalent queries to the original one. Two queries are defined to be *semantically equivalent* (Chu and Lee, 1990; Siegel, 1988) if they return identical answer given the same contents of the database. The reformulation from one query to another is by logical inference using *database abstractions*, the abstracted knowledge of the contents of relevant databases. The database abstractions describe the databases in terms of the set of closed formulas of first-order logic. These formulas describe the database in the sense that they are true with regard to all instances in the database. We define two classes of formulas: *range information* are propositions that assert the ranges of the values of database attributes; and *rules* are of the form of implications with an arbitrary number of range propositions on the antecedent side and one range proposition on the consequent side. Figure 4.3 shows a small set of the database abstractions. In all formulas the variables are implicitly universally quantified.

The first two rules in Figure 4.3 state that for all instances, the value of its attribute country name is `"GERMANY"` if and only if the value of its attribute country code is `"FRG"`. With these two rules, we can reformulate the subquery `SUBQ1` in Figure 4.4 to the equivalent subquery `SUBQ2` by replacing the constraint on `geo_geoloc.country_name` with the constraint on `geo_geoloc.country_code`. We can inversely reformulate `SUBQ2` to `SUBQ1` with the same rules. Given a subquery Q, let $C_1, \ldots, C_k$ be the set of range and interaction constraints in Q, the following *reformulation operators* return a semantically equivalent query:

- **Range Refinement:** A range-information proposition states that the values of an attribute A are within some range R_d. If a range constraint of A in Q constrains the values of A in some range R_i, then we can refine this range constraint by replacing the constraining range R_i with $R_i \cap R_d$.
- **Constraint Addition:** Given a rule $A \rightarrow B$, if Q implies A then we can add constraint B to Q.
- **Constraint Deletion:** Given a rule $A \rightarrow B$, and Q implies A. If there exists C_i in Q and B implies C_i, then we can delete C_i from Q.

- **Subquery Refutation:** Given a rule $A \rightarrow B$, and Q implies A, if there exists C_i in Q and B implies $\neg C_i$, then we can assert that Q will return **NIL**.

Replacing constraints is treated as a combination of addition and deletion. Note that these reformulation operators do not always lead to more efficient versions of the subquery. Knowledge about the access cost of attributes is required to guide the search. For example, suppose the only index is placed on the attribute **geo_geoloc.country_name**, then reformulate SUBQ2 to SUBQ1 will reduce the cost from $O(n)$ to $O(k)$, where n is the size of the database and k is the amount of data retrieved. However, if either **geo_geoloc.country_name** and **geo_geoloc.country_code** are not indexed, then we will prefer the lower cost short string attribute **geo_geoloc.country_code**. In this case, reformulating SUBQ1 to SUBQ2 becomes more reasonable. Figure 4.5 shows our subquery reformulation algorithm. We explain the algorithm below by showing how **SUBQ-REFORMULATION** reformulates the subquery SUBQ1, the lower query in the query plan in Figure 4.2.

Range Information:

```
1:(geo_geoloc.country_name ∈
 ("Taiwan" "Italy" "Denmark" "Germany"
  "Turkey"))
2:(afsc_port.geocode ∈
 ("BSRL" "HNTS" "FGTW" "VXTY" "WPKZ" "XJCS"))
3:(0 ≤ afsc_port.refrig_storage ≤ 1000)
```

Rules:

```
1:(geo_geoloc.country_name = "GERMANY")
  ⟹ (geo_geoloc.country_code = "FRG")
2:(geo_geoloc.country_code = "FRG")
  ⟹ (geo_geoloc.country_name = "Germany")
3:(geo_geoloc.country_code = "FRG")
  ⟹ (47.15 ≤ geo_geoloc.latitude ≤ 54.74)
4:(afsc_port.rail = "Y" )
  ⟹ (afsc_port.geocode ∈
   ("BSRL" "HNTS" "FGTW"))
5:(6.42 ≤ geo_geoloc.longitude ≤ 15.00)
  ∧ (47.15 ≤ geo_geoloc.latitude ≤ 54.74)
  ⟹ (geo_geoloc.country_code = "FRG")
```

Figure 4.3: Example of Database Abstractions

There are three input arguments in this algorithm. The first argument **Subquery** is the subquery to be reformulated. Another argument **DB-Knowledge** contains the set of range information and rules that describe the database queried by the input subquery. And **Cost-Model** contains the knowledge to decide the execution cost of constraints. Initially, all the range constraints are refined by applying the **range refinement** operator. The reason why we want to refine the constraining ranges is to make the subquery more likely to match many rules. This is because after range refinement, the constraining ranges are smaller and more likely to imply the antecedent of a rule. Range refinement also reduces comparisons in evaluating constraints on string type attributes. The only range constraint in SUBQ1 is on **geo_geoloc.country_name**, and its constrained value **GERMANY** is within the range of possible values (see the first formula of range information). Thus, this constraint is unchanged.

The second step is to match all applicable rules from the set of database abstractions using the reformulation operators defined above. If a **Subquery Refutation** rule is found then the subquery is refuted and the algorithm halts immediately. When a **Constraint Deletion** rule is found, then some constraints in the subquery are redundant and can be deleted from the subquery without changing the semantics. We only put the constraint in the **Inferred-Set** instead of actually deleting it from the subquery. This is because, its redundancy is due to the *logical* reason, not the performance consideration. More knowledge and analysis is required to decide whether it should be actually deleted. In the case that a **Constraint Addition** rule is

```
SUBQ1:
(retrieve (?geoloc ?geocode2)
 (:and (geo_geoloc?geoloc)
       (geo_geoloc.geocode ?geoloc ?geocode2)
       (geo_geoloc.country_name ?geoloc
        "Germany")))
SUBQ2:
(retrieve (?geoloc ?geocode2)
 (:and (geo_geoloc?geoloc)
       (geo_geoloc.geocode ?geoloc ?geocode2)
       (geo_geoloc.country_code ?geoloc
        "FRG")))
SUBQ3:
(retrieve (?geoloc ?geocode2)
 (:and (geo_geoloc?geoloc)
       (geo_geoloc.geocode ?geoloc ?geocode2)
       (geo_geoloc.country_code ?geoloc "FRG")
       (geo_geoloc.latitude ?geoloc ?latitude)
       (>= ?latitude 47.15)
       (<= ?latitude 54.74)))
```

Figure 4.4: Equivalent Subqueries

```
SUBQ-REFORMULATION(Subquery, DB-Knowledge, Cost-Model)
 1.refine range constraints, if Subquery refuted, return  Nil;
 2.for all applicable rules A  →  B in DB-Knowledge:
    if Subquery refuted, return  NIL;
    else add B to Inferred-Set, add (B,A) to Dependency-List;
 3.for all B in Inferred-Set in the order of their cost:
    if B is not indexed and ∃ (B,A) in Dependency-List
       delete B from Subquery, delete (B,A) from Dependency-List;
       replace all (C,B) in dependency list with (C,A);
 4.return (reformulated Subquery, Inferred-Set)
END.
```

Figure 4.5: Subquery Reformulation Algorithm

found, we add the constraint to the subquery and also put it in the **Inferred-Set**. The first rule in Figure 4.3 is matched and fired for SUBQ1 and we get an additional constraint (**geo_geoloc.country_code ?geoloc "FRG"**), which is added to the **Inferred-Set**. Then the second and third rules are matched because of the additional constraint on country code. The constraints **geo_geoloc.latitude** and **geo_geoloc.country_name** are added to the **Inferred-Set**.

The third step is to select the constraints in **Inferred-Set** to delete from the subquery. The selection is based on the constraint's relative estimated execution cost which is computed by the type of the constraints (range constraint, or interaction constraint), the type of the attribute's values (integer, string, and their length), and whether they are indexed. The information required for this estimation is available from the input **cost-model** provided by SIMS. The constraints in the **Inferred-Set** are sorted into the partial order of their cost and then deleted in this order until the total cost of the remaining constraints is less than the original subquery. To preserve the semantics of the subquery, we keep a dependency list of the inferred constraints to avoid deleting all constraints in an implication cycle. In our example, the attribute **geo_geoloc.country_name** is deleted because its long string type is the most expensive. The next most expensive constraint is the one on attribute **geo_geoloc.country_code**. However, it should be preserved because the cause of its deletability (i.e., the constraint on **geo_geoloc.country_name**) was just deleted. Finally, the constraint on **geo_geoloc.latitude** is kept because it is an indexed attribute that will improve the efficiency of the subquery. The algorithm returns the reformulated subquery SUBQ3 as shown in Figure 4.4, as well as the **Inferred-Set**, which will be used for reformulating the succeeding subqueries

in the query plan.

Not all rules are matched directly from the database abstractions. For interaction constraints, we have axioms for set inclusion and mathematical relations. For example, if there is an interaction constraint (> ?Y ?X) and we have rules or range information which assert that (> ?X 17), then we can add a new constraint (> ?Y 17) because (> ?X 17) $\wedge$ (> ?Y ?X) $\Rightarrow$ (> ?Y 17). These axioms are implemented as inference procedures for efficiency.

The worst case complexity of **SUBQ-REFORMULATION** is $O(R^2N*max(M, \log N))$, where M is the maximum length of the antecedent of the rules, N is the greatest number of constraints in the partially reformulated query, that is, the number of original constraints plus the number of added constraints in **Inferred-Set** before final selection, and R is the size of **DB-Knowledge**. The worst case cost to match a rule is $O(MN)$. Suppose the system matches applicable rules linearly in the set of the database abstractions, all rules must be matched and this takes RMN. The complexity of Step 2 is thus $O(R^2MN)$, in the case that only one rule is fired in every scan of the database abstractions, and every rule is eventually fired. The complexity of Step 3 is $O(N \log N)$, the cost of sorting. Deleting constraints in the **Inferred-Set** in their order of estimated cost takes $O(N)$.

Because the added constraints are range constraints of an attribute, the number of constraints will not exceed the number of the attributes of the relevant database tables.[1] Therefore, N is small compared to R. In the average case, the rule match cost is about $O(N)$, since the lengths of rules are usually less than 3. The database abstractions are normally scanned less than 3 times. Therefore, R dominates the complexity of the algorithm. With small values of R, this algorithm will not introduce significant overhead to the cost of query processing. To alleviate the impact of a large R on the system's performance, we can adopt sophisticated indexing and hashing techniques in rule matching, or restrain the size of the database abstractions by removing database abstractions with low utility.

```
QPLAN-REFORMULATION(Plan, DB-Knowledge, Cost-Model)
 1.KB ← DB-Knowledge;
 2.for all subqueries S in the order specified in Plan:
   (S',Inferred-Set) ←
SUBQ-REFORMULATION(S,KB,Cost-Model);
   if S' refuted, return  Nil;
   else update KB with Inferred-Set; update Plan with S';
 3.for all subqueries S whose semantics are changed:
   SUBQ-REFORMULATION(S, DB-Knowledge, Cost-Model);
 4.return reformulated Plan
END.
```

Figure 4.6: Query Plan Reformulation Algorithm

4 Query Plan Reformulation

We can reformulate each subquery in the query plan with the subquery reformulation algorithm and improve their efficiency. However, the most expensive aspect of the multidatabase query is often processing intermediate data. In the example query plan in Figure 4.2, the constraint on the final subqueries involves the variables **?geocode** and **?geocode2** that are bound in the preceding subqueries. If we can reformulate these preceding subqueries so that they retrieve only the data instances possibly satisfying the constraint **(= ?geocode ?geocode2)** in the final subquery, the intermediate data will be reduced. This requires the query plan reformulation algorithm to be able to propagate the constraints along the data flow paths in the query plan. The query plan reformulation algorithm defined in Figure 4.6 achieves this by updating the database abstractions and rearranging constraints. We explain the algorithm below using the query plan in Figure 4.2.

[1] If there are two constraints on the same attribute, we can always apply the **range refinement** operator on them and merge them together.

The algorithm takes three input arguments. The argument `Plan` is the input query plan, `DB-Knowledge` and `Cost-Model` are defined as in `SUBQ-REFORMULATION`. After the initialization step, in the second step, the algorithm reformulates each subquery in the partial order (i.e., the data flow order) specified in the plan. The two `db-retrieve` subqueries are reformulated first. The database abstractions are updated with `Inferred-Set` which is returned from `SUBQ-REFORMULATION` to propagate the constraints to later subqueries. For example, when reformulating the upper subquery, the fourth rule is fired for adding the constraint on the variable `?geocode` which is bound to the attribute `afsc_port.geocode`. Although this long string type constraint is then selected to be deleted, it reveals the range of `afsc_port.geocode` in the output data of the upper subquery. This range information together with other inferred constraints in `Inferred-Set` replaces the original range information to update the initial database abstractions. In this example, the second formula of the initial range information is replaced by (`afsc_port.geocode` $\in$ (`"BSRL" "HNTS" "FGTW"`)), the consequent condition of the fourth rule. The algorithm uses this updated range information to reformulate the final subquery and reduces the possible values from six to three. In addition, the constraint (`afsc_port.rail ?port "Y"`) in the upper subquery is propagated along the data flow path to its succeeding subquery implicitly.

Now that the updated range information for `?geocode` is available, the subquery reformulation algorithm can infer from the constraint (`= ?geocode ?geocode2`) a new constraint (`member ?geocode2 ("BSRL" "HNTS" "FGTW")`) and add it to the final subquery. However, this constraint should be executed by the remote DBMS instead of by the local Loom system, because it does not involve interaction with different databases. In this case, when updating the query plan with the reformulated subquery, the algorithm locates where the constrained variable of each new constraint is bound, and inserts the new constraint in the corresponding subqueries. In our example, the variable is bound by (`geo_geoloc.geocode ?geoloc ?geocode2`) in the lower subquery in Figure 4.2. The algorithm will insert the new constraint on `?geocode2` in that subquery. In this way, the constraints (`afsc_port.rail ?port "Y"`) and (`= ?geocode ?geocode2`) are propagated back along the data flow path to the lower subquery. This process of new constraint insertion is referred to as *constraint rearrangement*.

However, the semantics of the rearranged subqueries, such as the lower subquery in this example, are changed because of the newly inserted constraints. (Note, that the semantics of the overall query plan remain the same.) After all the subqueries in the plan have been reformulated, Step 3 of the algorithm reformulates these subqueries again to improve their efficiency. In our example, the reformulation algorithm is applied again to the lower subquery, but no reformulation is found to be appropriate. The final reformulated query plan is shown in Figure 4.7.

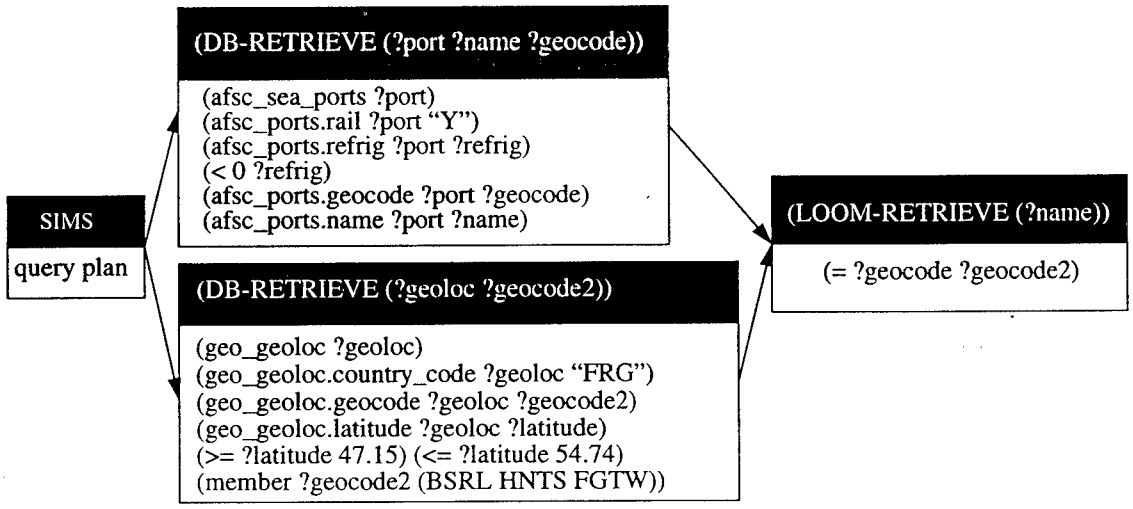


Figure 4.7: Reformulated SIMS Plan for Example Query

This query plan is more efficient and returns the same answer as the original one. In our example, the

lower subquery is more efficient because of the new constraint on the indexed attribute `geo_geoloc.latitude` (by `SUBQ-REFORMULATION`). The intermediate data items are reduced because of the new constraint on the attribute `geo_geoloc.geocode`. The logical rationale of this new constraint is derived from the constraints in the other two subqueries: `(afsc_port.rail ?port "Y")` and `(= ?geocode ?geocode2)`, and the fourth rule in the database abstractions.

The worst case complexity of `QPLAN-REFORMULATION` is $O(SR^2N*max(M,\log N))$, where S is the number of subqueries in the query plan, and $R^2N * max(M,\log N)$ is the cost of `SUBQ-REFORMULATION`. In the average case, S is less than 5, so the dominating factor is still the cost of the subquery reformulation $R^2N * max(M,\log N)$, in which R is the most important factor. Consequently, if R is relatively small, or we can match rules efficiently, this algorithm is efficient enough to be neglected in the total cost of query processing.

5 Experimental Results

The reformulation algorithms are implemented in the context of the SIMS system. which, for the purpose of our experiments, is connected with two distributed Oracle databases. Table 4.1 shows the size of these databases. The queries used were selected from the set of SQL queries constructed by other users of the databases. Table 4.2 lists the features of these queries. The first three queries are single database queries. The remaining queries access both databases, so they have two database subqueries and one subquery for evaluating interaction constraints and performing joins in Loom. The number of constraints includes the number of range and interaction constraints. The number of answers may not equal the number of retrieved instances, because the answers are results of projection on specified attributes and all duplicates are removed. Query 3 and 6 are null queries.

Database	Contents	Instances	Size(MByte)
Geo	Geographical locations	56708 rows in 16 tables	10.48
Assets	Planes,ships other assets	5728 rows in 14 tables	0.51

Table 4.1: Database Size

The performance statistics are shown in Table 4.3. All entries are based on an average of 10 trial executions. The number of rules fired counts both range information and rules used in reformulation. Note that a rule may be fired twice or more in Step 2 and 3 of the `QPLAN-REFORMULATION` algorithm. The amount of intermediate data indicated for each multidatabase query is the total number of the data instances retrieved from both databases and transferred to the SIMS system.

Query (short descriptions)	Database Accessed	Number of Subqueries	Number of Constraints	Number of Answers
1:Airports: runway≥8000, concrete surface	Geo	1	2	2
2:Locations: location code in state gsa code "TW"	Geo	1	2	147
3:Wharves: container cranes and rail track	Geo	1	4	0
4:Wharves: container/breakbulk ships	Geo,Assets	3	10	6
5:Ports: accommodate ship with code "1240"	Geo,Assets	3	4	2
6:Ports: accommodate ship "1207", mob "10C"	Geo,Assets	3	4	0
7:Ships: dock in channels of port in Long Beach	Geo,Assets	3	3	28
8:Ports & Ships: berths storage > ship capacity	Geo,Assets	3	1	9
9:Ports & Ships:ship length, fit berth type "TE"	Geo,Assets	3	4	20
10:Ports & Ships:Tunisia ports,frozen cargo unload	Geo,Assets	3	5	29

Table 4.2: Experiment Multidatabase Queries

The most noticeable cost reduction is achieved by reformulation when the system can determine the

answers of queries from its knowledge. In these queries, the system can eliminate the corresponding database access. For example, the system refutes Query 3 and 6 and returns the answer `NIL` immediately. In Query 7, the system asserts the answer of a database subquery. This subquery is eliminated, and the query reduces to a single database query. Query 8, 9, and 10 are typical multidatabase queries, the system reformulates them and eliminates a large amount of intermediate data. Query execution time is thus reduced by about a factor of 2. Query 2 is an expensive single database query. The system reformulates it by introducing a constraint on an indexed attribute and saves a considerable amount of time.

query	1	2	3	4	5	6	7	8	9	10
planning time (sec)	0.5	0.3	0.6	2.1	1.1	0.7	0.7	0.5	0.5	0.8
reformulation time	0.1	0.1	0.0	0.5	0.1	0.0	0.0	0.1	0.1	0.3
rules fired (times)	37	18	11	126	63	8	17	15	19	71
intermediate data w/o Ref[a]	-	-	-	145	41	1	810	956	808	810
intermediate data w/ Ref[b]	-	-	-	145	35	0	28	233	320	607
query execution time w/o Ref	0.3	8.2	0.6	12.3	11.3	2.0	251.0	401.8	255.8	258.8
query execution time w/ Ref	0.3	1.5	0.0	11.3	11.1	0.0	0.3	207.5	102.9	195.2
total elapsed time w/o Ref	0.8	8.5	1.2	14.4	12.4	2.7	251.7	402.3	256.3	259.6
total elapsed time w/ Ref	0.9	1.9	0.6	13.9	12.3	0.7	1.0	208.1	103.5	196.3

[a] w/o Ref = Without reformulation.

[b] w/ Ref = With reformulation.

Table 4.3: Experimental Results

There are cases where the reformulation did not achieve significant cost reduction. The first case is when the query is already very efficient. For example, in Query 1, the query execution time without reformulation is very short, and reformulation appears to be unnecessary. Another case is when the system can not reduce the amount of intermediate data, as in Query 4 and 5. This is due to a lack of sufficient database abstractions, or it may just be impossible to reduce the cost for some particular queries and databases. However, as indicated in the experimental results, the reformulation time is so short that even when no significant cost reduction can be achieved, the overhead will not degrade the performance of retrieval. To sum up, the reformulation approach is effective and can achieve a substantial cost reduction.

In this experiment, the system uses a set of database abstractions consisting of 203 rules about range information and 64 implication rules for every query plan. These database abstractions were prepared by compiling the databases. For range information, the compiling procedure summarizes the range of each attribute of the database by extracting the minimum and maximum values for numerical attributes, and enumerating the possible values for string type attributes. If the number of possible values exceeds a threshold, this range information is discarded. The implication rules were prepared by a semi-automatic learning algorithm similar to the KID3 (Piatetsky-Shapiro, 1991). This algorithm takes the user input condition A, and learns a set of rules of the form $A \rightarrow B$ from the database. The algorithm retrieves the data that satisfy the condition A, then compiles the data for the conclusions B.

6 Related Work

The semantic query optimization approach has been studied extensively in previous work (Chakravarthy, Grant, and Minker, 1990; Siegel, 1988; King, 1981; Hammer and Zdonik, 1980). These systems demonstrate the benefit of using knowledge of database contents to optimize queries. The most significant difference between our approach and theirs is that they rely on heuristics, and search for the optimal equivalent query in a hill-climbing manner, while our approach adopts a *delayed-commitment* strategy. Their systems search for the optimal query in the space of equivalent queries of the given query. Whenever a rule is fired, their systems will generate a new equivalent query, until an optimal one is found. This leads to a combinatorial explosion of equivalent queries among which the system needs to select. To overcome this problem, they

use heuristics and hill-climbing to prune the search space, but as a consequence, the reformulated query is usually only locally optimal. Sometimes, this process causes infinite loops that require more heuristics to resolve (Siegel, 1988).

To illustrate the problem of previous work, consider the following situation. Suppose there are two rules in the set of database abstractions, $A \rightarrow B$, and $B \rightarrow C$. Suppose we are given a query Q which implies A, and the rule $A \rightarrow B$ is the only applicable rule. Rule $B \rightarrow C$ will be applicable if B is added to Q by firing $A \rightarrow B$. Suppose further that Q and C are contradictory, and B is a costly constraint. For hill-climbing systems, $A \rightarrow B$ will never be fired since adding B will increase the cost. Thus, $B \rightarrow C$ will not be applicable. As a result, the system can not figure out that the answer of the query is null, unless it can backtrack. But backtracking requires the system to maintain a large set of equivalent queries. This overhead will make the system impractical.

In contrast, our subquery reformulation algorithm does not generate queries each time a rule is fired. Instead, we fire all applicable rules at once and collect the candidate constraints in a list `Inferred-Set` and then select only those that will contribute to the cost reduction. In the example above, our algorithm can consider both rules and refute the query without maintaining a large set of equivalent queries. This approach is a delayed-commitment strategy because the system delays the reformulation until it has enough information to make a decision. Although the algorithm fires all applicable rules, it is still polynomial. The empirical results show that it is efficient. Moreover, it is flexible because no additional specific heuristics are required. The list `Inferred-Set` turns out to be the information needed to propagate constraints among subqueries. Subsequently, the subquery reformulation is easily extended to query plan reformulation.

Compared to conventional syntactical optimization techniques for distributed database systems, our approach differs in both the knowledge brought to bear and the way queries are optimized. (Apers, Hevner, and Yao, 1983; Jarke and Koch, 1984; Ullman, 1988) describe approaches that use the *semi-join* operation to join two database relations in distributed databases. The semi-join techniques propagate constraints by computing a semi-join before performing the actual join. Our approach propagates constraints from knowledge of database abstractions without accessing remote databases, and thus has less overhead then the semi-join. The semi-join techniques may reduce intermediate data when the result of semi-join is significantly smaller than the entire relevant relations. However, there are situations when semi-join degrades the performance. The system needs to know the *reduction factors* of each semi-join to decide a semi-join schedule that will save execution cost. To compute reduction factor requires knowledge of the size of relevant relations and their joining path. It is usually difficult to estimate the size of an intermediate relation when the query is complicated. Some semi-join approaches assume a unrealistically simplified model to reduce the overhead, but to make semi-join approach effective, the system still need to bring to bear extensive statistical knowledge to estimate relation sizes (Jarke and Koch, 1984).

Another difference between our approach and the conventional distributed query optimization techniques is that they assume a homogeneous environment. They can transfer data from one site to another without any transformation. They can also distribute a relation into fragments and store them in different sites. Data distribution strategy and execution order scheduling are their major concerns. We assume a heterogeneous environment, so we focus on flexible reformulation on the semantic aspects of queries. In the future, we also intend to include size and system configuration information in our planning and reformulation algorithm to optimize query plans on the execution order.

7 Conclusion

This chapter presented a problem reformulation approach to reducing the cost of domain-modeled multi-database queries. The reformulation is based on logical inferences from database abstractions. This simple, efficient algorithm reduces the cost of the query plan by reducing intermediate data and refining each subquery. This is achieved without database implementation dependent heuristic control. Empirical results demonstrate that this algorithm can provide significant reductions in the cost of executing query plans.

One of the limitations of our implementation is that the rule match algorithm is linear to the size of the database abstractions. A very large set of database abstractions could make the reformulation costly. To avoid this problem, we plan to adopt a more sophisticated rule match algorithm, such as the RETE algorithm (Forgy, 1982), or its more efficient variations, to reduce this impact.

One important issue not addressed in this chapter is how to automatically acquire the database abstractions for reformulation (Siegel, 1988). We are now developing a learning algorithm that is driven by example queries (Hsu and Knoblock, 1993). We plan to use inductive learning (Cai, Cercone, and Han 1991; Haussler, 1988; Michalski, 1983) to identify the costly aspects of the example subqueries and propose candidate rules. The candidate rules will then be refined and learned by the system. After the system has learned a set of database abstractions, it needs to monitor their utility and validity to maintain the system's performance. We will address this issue in the future work.

Chapter 5

Rule Induction for Semantic Query Optimization

1 Introduction

Speeding up a system's performance is one of the major goals of machine learning. Explanation-based learning is typically used for speedup learning, while applications of inductive learning are usually limited to data classifiers. In this chapter, we present an approach in which inductively learned knowledge is used for semantic query optimization to speed up query answering for data/knowledge-based systems.

The principle of semantic query optimization [27] is to use semantic rules. such as *all Tunisian seaports have railroad access*, to reformulate a query into a less expensive but equivalent query, so as to reduce the query evaluation cost. For example, suppose we have a query to *find all Tunisian seaports with railroad access and 2,000,000 ft^3 of storage space.* From the rule given above, we can reformulate the query so that there is no need to check the railroad access of seaports, which may save some execution time. Many algorithms for semantic query optimization have been developed [22, 27, 48, 50]. Average speedup ratios from 20 to 40 percent using hand-coded knowledge are reported in the literature. This approach to query optimization has gained increasing attention recently because it is applicable to almost all existing data/knowledge-base systems. This feature makes it particularly suitable for intelligent information servers connected to various types of remote information sources.

A critical issue of semantic query optimization is how to encode useful background knowledge for reformulation. Most of the previous work in semantic query optimization in the database community assume that the knowledge is given. [27] proposed using *semantic integrity constraints* for reformulation to address the knowledge acquisition problem. Examples of semantic integrity constraints are *The salary of an employee is always less than his manager's*, and *Only female patients can be pregnant.* However, the integrity rules do not reflect properties of the contents of databases, such as related size of conceptual units, cardinality and distribution of attribute values. These properties determine the execution cost of a query. Moreover, integrity constraints rarely match query usage patterns. It is difficult to manually encode semantic rules that both reflect cost factors and match query usage patterns. The approach presented in this chapter uses example queries to trigger the learning in order to match query usage patterns and uses an inductive learning algorithm to derive rules that reflect the actual contents of databases.

An important feature of our learning approach is that the inductive algorithm learns from complex real-world information sources. In these information sources, data objects are clustered into conceptual units. For example, the conceptual unit of a relational databases is a *relation*, or simply a *table*. For object-based databases, it is a *class*. In description-logic knowledge bases [9], data instances are clustered into *concepts*. Each conceptual unit has attributes that describe the relevant features. Most inductive learning systems, such as ID3, assume that relevant attributes are given. Consider a database with three relations: `car`, `person`, and `company`. We might want to characterize a class of persons by the company they work for, or by the cars they drive, or by the manufacturers of their cars. In these cases, we need attributes from different relations to describe a desired class of objects. Previous studies [2, 46] have shown that the choice

of *instance language bias* (i.e., selecting an appropriate set of attributes) is critical for the performance of an inductive learning system. To address this problem, we propose an inductive learning algorithm that can select attributes from different relations automatically.

The remainder of this chapter is organized as follows. The next section illustrates the problem of semantic query optimization for data/knowledge bases. Section 3 presents an overview of the learning approach. Section 4 describes our inductive learning algorithm for structural data/knowledge bases. Section 5 shows the experimental results of using learned knowledge in reformulation. Section 6 surveys related work. Section 7 reviews the contributions of the chapter and describes some future work.

2 Semantic Query Optimization

Semantic query optimization is applicable to various types of database and knowledge base. Nevertheless, we chose the relational model to describe our approach because it is widely used in practice. The approach can be easily extended to other data models. In this chapter, a *database* consists of a set of primitive relations. A *relation* is then a set of instances. Each *instance* is a vector of attribute values. The number of attributes is fixed for all instances in a relation. The values of attributes can be either a number or a symbol, but with a fixed type. Below is an example database with two relations and their attributes:

```
geoloc(name,glc_cd,country,latitude,longitude),
seaport(name,glc_cd,storage,silo,crane,rail).
```

where the relation `geoloc` stores data about geographic locations, and the attribute `glc_cd` is a geographic location code.

The queries we are considering here are Horn-clause queries. A query always begins with the predicate `answer` and has the desired information as argument variables. For example,

```
Q1: answer(?name):-
      geoloc(?name,?glc_cd,"Malta",_,_),
      seaport(_,?glc_cd,?storage,_,_,_),
      ?storage > 1500000.
```

retrieves all geographical location names in Malta. There are two types of literals. The first type corresponds to a relation stored in a database. The second type consists of built-in predicates, such as `>` and `member`. Sometimes they are referred to as *extensional* and *intentional* relations, respectively (see [56]). We do not consider negative literals and recursion in this chapter.

Semantic rules for query optimization are also expressed in terms of Horn-clause rules. Semantic rules must be consistent with the contents of a database. To clearly distinguish a rule from a query, we show queries using the Prolog syntax and semantic rules in a standard logic notation. A set of example rules are shown as follows:

```
R1: geoloc(_,_,"Malta",?latitude,_)
    ⇒ ?latitude ≥ 35.89.

R2: geoloc(_,?glc_cd,"Malta",_,_)
    ⇒ seaport(_,?glc_cd,_,_,_,_).

R3: seaport(_,?glc_cd,?storage,_,_,_) ∧
    geoloc(_,?glc_cd,"Malta",_,_)
    ⇒ ?storage > 2000000.
```

Rule `R1` states that the latitude of a Maltese geographic location is greater than or equal to 35.89. `R2` states that all Maltese geographic locations *in the database* are seaports. `R3` states that all Maltese seaports have storage capacity greater than 2,000,000 ft^3. Based on these rules, we can infer five equivalent queries of `Q1`. Three of them are:

```
Q21: answer(?name):-
       geoloc(?name,?glc_cd,"Malta",_,_),
       seaport(_,?glc_cd,_,_,_,_).

Q22: answer(?name):-
       geoloc(?name,_,"Malta",_,_).
```

```
Q23: answer(?name):-
       geoloc(?name,_,"Malta",?latitude,_),
       ?latitude ≤ 35.89.
```

Q21 is deduced from Q1 and R3. This is an example of *constraint deletion* reformulation. From R2, we can delete one more literal on **seaport** and infer that Q22 is also equivalent to Q1. In addition to deleting constraints, we can also add constraints to a query based on rules. For example, we can add a constraint on **?latitude** to Q22 from R1, and the resulting query Q23 is still equivalent to Q1. Sometimes, the system can infer that a query is unsatisfiable because it contradicts a rule (or a chain of rules). It is also possible for the system to infer the answer directly from the rules. In both cases, there is no need to access the database to answer the query, and we can achieve nearly 100 percent savings.

Now that the system can reformulate a query into equivalent queries based on the semantic rules, the next problem is how to select the equivalent query with the lowest cost. The shortest equivalent query is not always the least expensive. The exact execution cost of a query depends on the physical implementation and the contents of the data/knowledge bases. However, we can usually estimate an approximate cost from the database schema and relation sizes. In our example, assume that the relation **geoloc** is very large and is sorted only on **glc_cd**, and assume that the relation **seaport** is small. Executing the shortest query Q22 requires scanning the entire set of **geoloc** relations and is thus even more expensive than executing the query Q1. The cost to evaluate Q21 will be less expensive than Q1 and other equivalent queries because a redundant constraint on **?storage** is deleted, and the system can still use the sorted attribute **glc_cd** to locate the answers efficiently. Therefore, the system will select Q21.

Although the number of equivalent queries grows combinatorially with the number of applicable rules, semantic query optimization can be computed without explicitly searching this huge space. We have developed an efficient reformulation algorithm that is polynomial in terms of the number of applicable rules. We also extended this algorithm to reformulate multidatabase query access plans and showed that the reformulations produce substantial performance improvements [22].

We conclude this section with the following observations on semantic query optimization.

1. Semantic query optimization can reduce query execution cost substantially.

2. Semantic query optimization is not a tautological transformation from the given query; it requires nontrivial, domain-specific background knowledge. Learning useful background knowledge is critical.

3. Since the execution cost of a query is dependent on the properties of the contents of information sources being queried, the utility of a semantic rule is also dependent on these properties.

4. The overhead of reformulation is determined by the number of applicable rules. Therefore, the utility problem [35] is likely to arise and the learning must be selective.

3 Overview of the Learning Approach

This section presents an overview of our learning approach to address the knowledge acquisition problem of semantic query optimization. The key idea of our learning approach is that we view a query as a logical description (conjunction of constraints) of the answer, which is a set of instances satisfying the query. With an appropriate bias, an inductive learner can derive an equivalent query that is less expensive to evaluate than the original. Based on this idea, the learning is triggered by an example query that is expensive to evaluate. The system then inductively constructs a less expensive equivalent query from the data in the databases. Once this equivalent query is learned, the system compares the input query and constructed query, and infers a set of semantic rules for future use.

Figure 5.1 illustrates a simple scenario of this learning approach. An example query is given to a small database table with 3 instances. Evaluating this query will return an instance, which is marked with a ''+'' sign. Conceptually, instances in this table are labeled by the query as positive (answers) or negative (non-answers). We can use the inductive learning algorithm to generate an equivalent *alternative query* with appropriate biases so that the generated query is less expensive to evaluate. The generated alternative query should be satisfied by all answer instances and none of the others. This guarantees the equivalence of the two

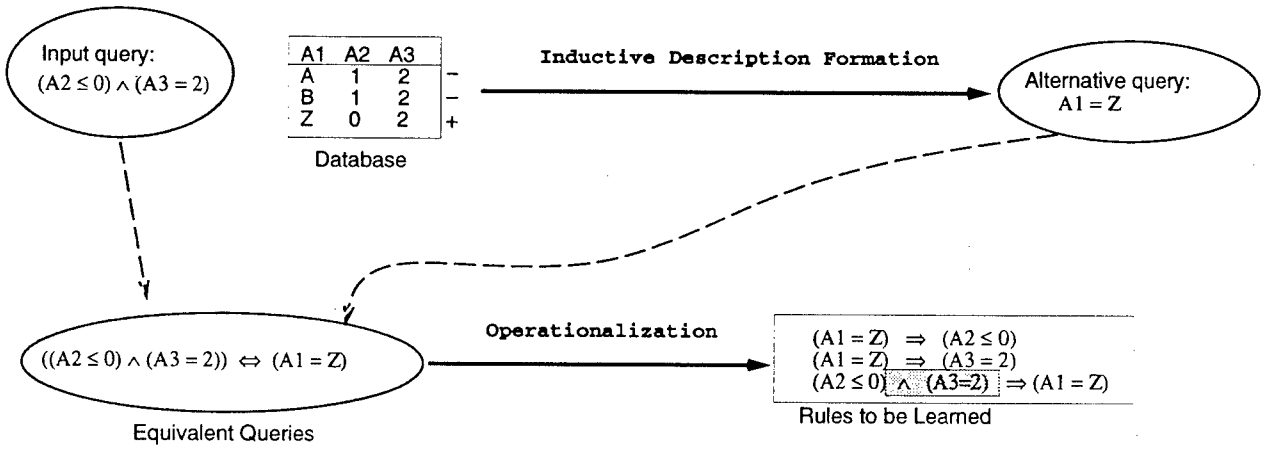


Figure 5.1: An Simplified Example Learning Scenario

queries with regard to the current status of the data/knowledge base. Suppose that in this simple database, a short query is always less expensive to execute. The system will bias the learning in favor of the shortest description and inductively learn an alternative query `(A1 = 'Z')`. The inductive learning algorithm will be discussed in the next section.

The equivalence of the alternative query and the example query provides a *training example* of reformulation. In other words, this training example shows which equivalent query an input query should be reformulated into. The operationalization component will deduce a set of rules from the training example. This process consists of two stages. In the first stage, the system uses a logical inference procedure to transform the training example into the required syntax (Horn clauses). This syntax is designed so that the query reformulation can be computed efficiently. The equivalence between the two queries is converted to two implication rules:

$$(1)(A2 \leq 0) \wedge (A3 = 2) \Longrightarrow (A1 = `Z')$$

$$(2)(A1 = `Z') \Longrightarrow (A2 \leq 0) \wedge (A3 = 2)$$

Rule (2) can be further expanded to satisfy our syntax criterion:

$$(3)(A1 = `Z') \Longrightarrow (A2 \leq 0)$$

$$(4)(A1 = `Z') \Longrightarrow (A3 = 2)$$

After the transformation, we have proposed rules (1), (3), and (4) that satisfy our syntax criterion. In the second stage, the system tries to compress the antecedents of rules to reduce their match costs. In our example, rules (3) and (4) contain only one literal as antecedent, so no further compression is necessary. These rules are then returned immediately and learned by the system.

If the proposed rule has more than one antecedent literal, such as rule (1), then the system can use the *greedy minimum set cover* algorithm [17] to eliminate unnecessary constraints. The problem of minimum set cover is to find a subset from a given collection of sets such that the union of the sets in the subset is equal to the union of all sets. We rewrite rule (1) as

$$(5)\neg(A1 = `Z') \Longrightarrow \neg(A2 \leq 0) \vee \neg(A3 = 2).$$

The problem of compressing rule (1) is thus reduced to the following: given a collection of sets of data instances that satisfy $\neg$`(A2 ≤ 0)` $\vee$ $\neg$`(A3 = 2)`, find the minimum number of sets that cover the set of data instances that satisfy $\neg$`(A1 = 'Z')`. Since the resulting minimum set that covers $\neg$`(A1 = 'Z')` is $\neg$`(A2 ≤ 0)`, we can eliminate $\neg$`(A3 = 2)` from rule (5) and negate both sides to form the rule

$$(A2 \leq 0) \Longrightarrow (A1 = `Z').$$

```
      geoloc("Safaqis",    8001, Tunisia, ...)     seaport("Marsaxlokk"   8003
...)
      geoloc("Valletta",   8002, Malta,   ...)+    seaport("Grand Harbor" 8002
...)
      geoloc("Marsaxlokk", 8003, Malta,   ...)+    seaport("Marsa"        8005
...)
      geoloc("San Pawl",   8004, Malta,   ...)+    seaport("St Pauls Bay" 8004
...)
      geoloc("Marsalforn", 8005, Malta,   ...)+    seaport("Catania"      8016
...)
      geoloc("Abano",      8006, Italy,   ...)     seaport("Palermo"      8012
...)
      geoloc("Torino",     8007, Italy,   ...)     seaport("Traparri"     8015
...)
      geoloc("Venezia",    8008, Italy,   ...)     seaport("AbuKamash"    8017
...)
                ⋮             ⋮
```

Figure 5.2: The Database Fragment

4 Learning Alternative Queries

The scenario shown in Figure 5.1 is a simplified example where the database consists of only one table. However, real-world databases and knowledge bases usually decompose their application domain into multiple conceptual units. One could try to combine every conceptual unit that could be relevant into a large table, then apply the learning system for tabular databases directly. However, learning from a large table is too expensive computationally. Such an approach will not work unless a small number of relevant attributes are correctly identified before learning.

In this section, we discuss inductive learning for Horn-clause queries from a database with multiple relations. Our learning problem is to find an alternative query to characterize a class of instances defined in a relation. In standard machine learning terms, this subset of instances are labeled as positive examples, and the others are negative examples.

Before we discuss the algorithm, we need to clarify two forms of constraints implicitly expressed in a query. One form is an *internal disjunction*, a set of disjunctions on the values of an attribute. For example, an instance of `geoloc` satisfies:

```
C1:
 geoloc(?name,_,?cty,_,_),
 member(?cty,["Tunisia","Italy","Libya"]).
```

iff its `?cty` value is `"Tunisia"`, `'"Italy"`, or `"Libya"`. The other form is a *join constraint*, which combines instances from two relations. For example, a pair of instances of `geoloc` and `seaport` satisfy a join constraint:

```
C2:geoloc(?name1,?glc_cd,_,_,_),
   seaport(?name2,?glc_cd,_,_,_,_).
```

iff they share common values on the attribute `glc_cd` (geographic location code).

Our inductive learning algorithm is extended from the greedy algorithm that learns internal disjunctions proposed by [20]. Of the many inductive learning algorithms, Haussler's was chosen because its hypothesis description language is the most similar to ours. His algorithm starts from an empty hypothesis of the target concept description to be learned. The algorithm proceeds by constructing a set of candidate constraints that are consistent with all positive examples, and then using a *gain/cost* ratio as the heuristic function to select and add candidates to the hypothesis. This process of candidate construction and selection is repeated until no negative instance satisfies the hypothesis.

We extended Haussler's algorithm to allow join constraints in the target description. To achieve this, we extended the candidate construction step to allow join constraints to be considered, and we extended the heuristic function to evaluate both internal disjunctions and join constraints. Also, we adopted an approach to searching the space of candidate constraints that restricts the size of the space.

4.1 Constructing and Evaluating Candidate Constraints

In this subsection, we describe how to construct a candidate constraint, which can be either an internal disjunction or a join constraint. Then we describe a method for evaluating both internal disjunctions and join constraints. Given a relation partitioned into positive and negative instances, we can construct an internal disjunctive constraint for each attribute by generalizing attribute values of positive instances. The constructed constraint is consistent with positive instances because it is satisfied by all positive instances. Similarly, we can construct a join constraint consistent with positive instances by testing whether all positive instances satisfy the join constraint. The constructed constraints are candidates to be selected by the system to form an alternative query.

For example, suppose we have a database that contains the instances as shown in Figure 5.2. In this database, instances labeled with ''+'' are positive instances. Suppose the system is testing whether join constraint C2 is consistent with the positive instances. Since for all positive instances, there is a corresponding instance in seaport with a common glc_cd value, the join constraint C2 is consistent and is considered as a candidate constraint.

Once we have constructed a set of candidate internal disjunctive constraints and join constraints, we need to measure which one is the most promising and add it to the hypothesis. In Haussler's algorithm, the measuring function is a *gain/cost* ratio, where *gain* is defined as the number of negative instances excluded and *cost* is defined as the syntactic length of a constraint. This heuristic is based on the generalized problem of minimum set cover where each set is assigned a constant cost. Haussler used this heuristic to bias the learning for short hypotheses. In our problem, we want the system to learn a query expression with the least cost. In real databases, sometimes additional constraints can reduce query evaluation cost. So we keep the *gain* part of the heuristic, while defining the *cost* of the function as the estimated evaluation cost of the constraint by a database system.

The motivation of this formula is also from the generalized minimum set covering problem. The gain/cost heuristic has been proved to generate a set cover within a small ratio bound ($\ln|n|+1$) of the optimal set covering cost [15], where n is the number of input sets. However, in this problem, the cost of a set is a constant and the total cost of the entire set covers is the sum of the cost of each set. This is not always the case for database query execution, where the cost of each constraint is dependent on the execution ordering. To estimate the actual cost of a constraint is very expensive. We therefore use an approximation heuristic here.

The evaluation cost of individual constraints can be estimated using standard database query optimization techniques [57] as follows. Let $\mathcal{D}_1$ denote the constraining relation, and $|\mathcal{D}_1|$ denote the size of a relation, then the evaluation cost for an internal disjunctive constraint is proportional to

$$|\mathcal{D}_1|$$

because for an internal disjunction on an attribute that is not indexed, a query evaluator has to scan the entire database to find all satisfying instances. If the internal disjunction is on an indexed attribute, then the cost should be proportional to the number of instances satisfying the constraint. In both cases, the system can always sample the database query evaluator to obtain accurate execution costs.

For join constraints, let $\mathcal{D}_2$ denote the new relations introduced by a join constraint, and $\mathcal{I}_1$, $\mathcal{I}_2$ denote the *cardinality* of join attributes of two relations, that is, the number of distinct values of attributes over which $\mathcal{D}_1$ and $\mathcal{D}_2$ join. Then the evaluation cost for the join over $\mathcal{D}_1$ and $\mathcal{D}_2$ is proportional to

$$|\mathcal{D}_1| * |\mathcal{D}_2|$$

when the join is over attributes that are not indexed, because the query evaluator must compute a cross product to locate pairs of satisfying instances. If the join is over indexed attributes, the evaluation cost is proportional to the number of instance pairs returned from the join, that is,

$$\frac{|\mathcal{D}_1| * |\mathcal{D}_2|}{max(\mathcal{I}_1, \mathcal{I}_2)}.$$

This estimate assumes that distinct attribute values distribute uniformly in instances of joined relations. Again, if possible, the system can sample the database for more accurate execution costs. For the above example problem, we have two candidate constraints that are the most promising:

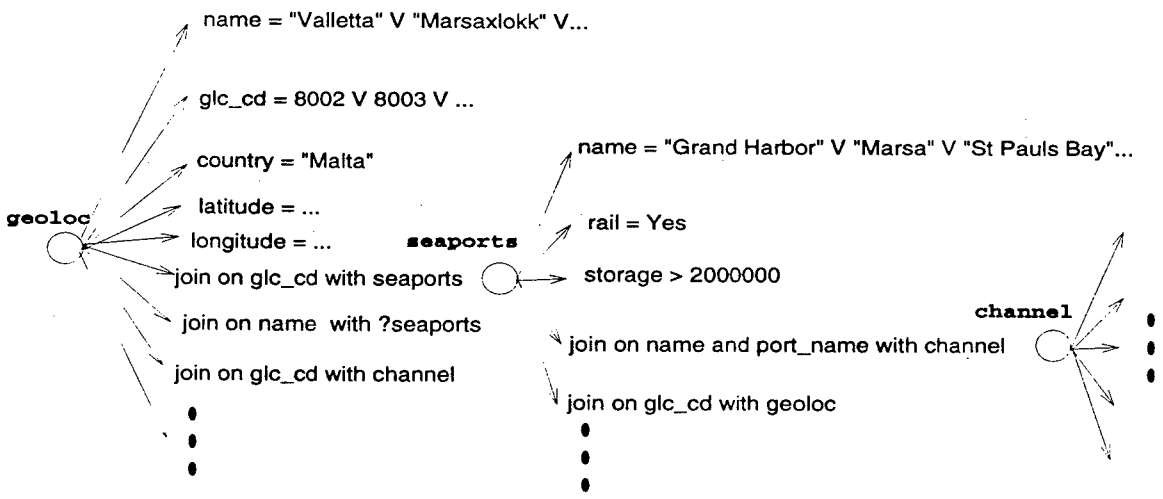


Figure 5.3: Candidate Constraints to be Selected

```
C3:geoloc(?name,_,"Malta",_,_).

C4:geoloc(?name,?glc_cd,_,_,_),
   seaport(_,?glc_cd,_,_,_,_).
```

Suppose |**geoloc**| is 300, and |**seaport**| is 8. Cardinality of **glc_cd** for **geoloc** is 300 again, and for **seaport** is 8. Suppose both relations have indices on **glc_cd**. Then the evaluation cost of **C3** is 300, and **C4** is $300 * 8/300 = 8$. The gain of **C3** is $300 - 4 = 296$, and the gain of **C4** is $300 - 8 = 292$, because only 4 instances satisfy **C3** while 8 instances satisfy **C4**. (There are 8 seaports, and all have a corresponding **geoloc** instance.) So the gain/cost ratio of **C3** is $296/300 = 0.98$, and the gain/cost ratio of **C4** is $292/8 = 36.50$. The system will select **C4** and add it to the hypothesis.

4.2 Searching the Space of Candidate Constraints

When a join constraint is selected; a new relation and its attributes are introduced to the search space of candidate constraints. The system can consider adding constraints on attributes of the newly introduced relation to the partially constructed hypothesis. In our example, a new relation **seaport** is introduced to describe the positive instances in **geoloc**. The search space is now expanded into two layers, as illustrated in Figure 5.3. The expanded constraints include a set of internal disjunctions on attributes of **seaport**, as well as join constraints from **seaport** to another relation. If a new join constraint has the maximum gain/cost ratio and is selected later, the search space will be expanded further. Figure 5.3 shows the situation when a new relation, say **channel**, is selected, the search space will be expanded one layer deeper. At this moment, candidate constraints will include all unselected internal disjunctions on attributes of **geoloc**, **seaport**, and **channel**, as well as all possible joins with new relations from **geoloc**, **seaport** and **channel**. Exhaustively evaluating the gain/cost of all candidate constraints is impractical when learning from a large and complex database.

We adopt a search method that favors candidate constraints on attributes of newly introduced relations. That is, when a join constraint is selected, the system will estimate only those candidate constraints in the newly expanded layer, until the system constructs a hypothesis that excludes all negative instances (i.e., reaches the goal) or no more consistent constraints in the layer with positive gain are found. In the later case, the system will backtrack to search the remaining constraints on previous layers. This search control bias takes advantage of underlying domain knowledge in the schema design of databases. A join constraint is unlikely to be selected on average, because an internal disjunction is usually much less expensive than a join. Once a join constraint (and thus a new relation) is selected, this is strong evidence that all useful internal disjunctions in the current layer have been selected, and it is more likely that useful candidate constraints are on attributes of newly joined relations. This bias works well in our experiments. But certainly there are

cases when this search heuristic prunes out useful candidate constraints. Another way to bias the search is by including prior knowledge for learning. In fact, it is quite natural to include prior knowledge in our algorithm, and we will discuss this later.

Returning to the example, since C4 was selected, the system will expand the search space by constructing consistent internal disjunctions and join constraints on **seaport**. Assuming that the system cannot find any candidate on **seaport** with positive gain, it will backtrack to consider constraints on **geoloc** again. Next, the constraint on **country** is selected (see Figure 5.3) and all negative instances are excluded. The system thus learns the query:

```
Q3: answer(?name):-
      geoloc(?name,?glc_cd,"Malta",_,_),
      seaport(_,?glc_cd,_,_,_,_).
```

The operationalization component will then take Q1 and this learned query Q3 as a training example for reformulation,

```
geoloc(?name,_,"Malta",_,_)
⇔ geoloc(?name,?glc_cd,"Malta",_,_) ∧
   seaport(_,?glc_cd,_,_,_,_).
```

and deduce a new rule to reformulate Q1 to Q3:

```
geoloc(_,?glc_cd,"Malta",_,_)
⇒ seaport(_,?glc_cd,_,_,_,_).
```

This is the rule R2 we have seen in Section 2. Since the size of **geoloc** is considerably larger than that of **seaport**. next time when a query asks about geographic locations in Malta, the system can reformulate the query to access the **seaport** relation instead and speed up the query answering process.

The algorithm can be further enhanced by including prior knowledge to reduce the search space. The idea is to use prior knowledge, such as *determinations* proposed by [46], to sort candidate constraints by their comparative relevance. and then test their gain/cost ratio in this sorted order. For example, assuming that from its prior knowledge the system knows that the constraints on attributes **latitude** and **longitude** of **geoloc** are unlikely to be relevant, then the system can ignore them and evaluate candidate constraints on the other attributes first. If the prior knowledge is correct, the system will construct a consistent hypothesis with irrelevant constraints being pruned from the search space. However, if the system cannot find a constraint that has a positive gain, then the prior knowledge may be wrong, and the system can backtrack to consider "irrelevant" constraints and try to construct a hypothesis from them. In this way, the system can tolerate incorrect and incomplete prior knowledge. This usage of prior knowledge follows the general spirit of FOCL [40].

5 Experimental Results

Our experiments are performed on the SIMS knowledge-based information server [4, 22]. SIMS allows users to access different kinds of remote databases and knowledge bases as if they were using a single system. For the purpose of our experiments, SIMS is connected with three remotely distributed Oracle databases via the Internet. Table 5.1 shows the domain of the contents and the sizes of these databases. We had 34 sample queries written by users of the databases for the experiments. We classified these queries into 8 categories according to the relations and constraints used in the queries. We then chose 8 queries randomly from each category as input to the learning system and generated 32 rules. These rules were used to reformulate the remaining 26 queries. In addition to learned rules, the system also used 163 attribute range facts (e.g., the range of the **storage** attribute of **seaport** is between 0 and 100,000) compiled from the databases. Range facts are useful for numerically typed attributes in the rule matching.

Table 5.1: Database Features

Databases	Contents	Relations	Instances	Size(MB)
Geo	Geographical locations	16	56708	10.48
Assets	Air and sea assets	14	5728	0.51
Fmlib	Force module library	8	3528	1.05

The performance statistics for query reformulation are shown in Table 5.2. In the first column, we show the average performance of all tested queries. We divide the queries into 3 groups. The number of queries in each group is shown in the first row. The first group contains those unsatisfiable queries refuted by the learned knowledge. In these cases, the reformulation takes full advantage of the learned knowledge and the system does not need to access the databases at all, so we separate them from the other cases. The second group contains those low-cost queries that take less than one minute to evaluate without reformulation. The last group contains the high-cost queries.

The second row lists the average elapsed time of query execution without reformulation. The third row shows the average elapsed time of reformulation and execution. Elapsed time is the total query processing time, from receiving a query to displaying all answers. To reduce inaccuracy due to the random latency time in network transmission, all elapsed time data are obtained by executing each query 10 times and then computing the average. The reformulation yields significant cost reduction for high-cost queries. The overall average gain is 57.10 percent, which is better than systems using hand-coded rules for semantic optimization [22, 48, 50]. The gains are not so high for the low-cost group. This is not unexpected, because the queries in this group are already very cheap and the cost cannot be reduced much further. The average overheads listed in the table show the time in seconds used in reformulation. This overhead is very small compared to the total query processing time. On average, the system fires rules 5 times for reformulation. Note that the same rule may be fired more than once during the reformulation procedure (see (Hsu & Knoblock 1993) for more detailed descriptions).

Table 5.2: Performance Statistics

	All	Answer inferred	$\leq$ 60s.	$>$ 60s.
# of queries	26	4	17	5
No reformulation	54.27	44.58	10.11	212.21
Reformulation	23.28	5.45	8.79	86.78
Time saved	30.99	39.14	1.31	125.46
% Gain of total elapsed time	57.1%	87.8%	12.9%	59.1%
Average overhead	0.08	0.07	0.07	0.11
Times rule fired	5.00	6.00	4.18	7.00

6 Related Work

Previously, two systems that learn background knowledge for semantic query optimization were proposed by [51] and by [47]. Siegel's system uses predefined heuristics to drive learning by an example query. This approach is limited because the heuristics are unlikely to be comprehensive enough to detect missing rules for various queries and databases. Shekhar's system is a data-driven approach which assumes that a set of relevant attributes is given. Focusing on these relevant attributes, their system explores the contents of the database and generates a set of rules in the hope that all useful rules are learned. Siegel's system goes to one extreme by neglecting the importance of guiding the learning according to the contents of databases, while Shekhar's system goes to another extreme by neglecting dynamic query usage patterns. Our approach is more flexible because it addresses both aspects by using example queries to trigger the learning and using inductive learning over the contents of databases for semantic rules.

The problem of inductive learning from a database with multiple relations shares many issues with research work in inductive logic programming (ILP) (Muggleton et al. 1994), especially the issue of when to introduce new relations. The main difference between our approach and ILP is that we also consider the cost of the learned concept description. Our system currently learns only single-clause, non-recursive queries, while ILP approaches can learn multi-clause and recursive rules. However, due to the complexity of the problem, most of the existing ILP approaches do not scale up well to learn from large, real-world data/knowledge-bases containing more than ten relations with thousands of instances. Our approach can learn from large databases because it also uses the knowledge underlying the database design.

Tan's cost-sensitive learning [52] is an inductive learning algorithm that also takes the cost of the learned description into account. His algorithm tries to learn minimum-cost decision trees from examples in a robot object-recognition domain. The algorithm selects a minimum number of attributes to construct a decision tree for recognition. The attributes are selected in the order of their evaluation cost. When constructing a decision tree, it uses a heuristic attribute selection function I^2/C, where I is the information gain defined as in ID3, and C is the cost to evaluate a given attribute. This function is similar to our function *gain/evaluation_cost*. While there is no theoretic analysis about the general performance of the heuristic I^2/C for decision-tree learning, our function is derived from approximation heuristics for minimum set cover problems. [38] defined another similar heuristic $(2^I - 1)/C$ for cost-sensitive decision-tree learning. His paper provides an information-theoretic motivation of the heuristic.

In [10] they present an attribute-oriented learning approach designed to learn from relational databases. The approach learns conjunctive rules by generalizing instances of a single relation. The generalization operations include replacing attribute values with the least common ancestors in a value hierarchy, removing inconsistent attributes, and removing duplicate instances. In contrast to our inductive learning algorithm, this attribute-oriented approach requires users to select relevant attributes before learning can be performed.

The operationalization component in our learning approach can be enhanced with an EBL-like explainer to filter out low utility rules and generalize rules. A similar "induction-first then EBL" approach can be found in [49]. Shen's system uses general heuristics to guide the inductive learning for regularities expressed in a rule template $P(x, y) \wedge R(y, z) \Rightarrow Q(x, z)$. Our system has a definite goal, so we use example queries to guide the learning and do not restrict the format of learned rules to a specific template.

7 Conclusions and Future Work

This chapter demonstrates that the knowledge required for semantic query optimization can be learned inductively under the guidance of example queries. We have described a general approach in which inductive learning is triggered by example queries, and an algorithm to learn from a database with multiple relations. Experimental results show that query reformulation using learned background knowledge produces substantial cost reductions for a real-world intelligent information server.

In future work, we plan to experiment with different ways of selecting example queries for training, and to develop an effective approach to using prior knowledge for constraining searches in the inductive learning algorithm. We also plan to enhance the operationalization component so that the system can be more selective and thus avoid the utility problem.

A limitation to our approach is that there is no mechanism to deal with changes to data/knowledge bases. There are three possible alternatives to address this problem. First, the system can simply remove the invalid rules due to the update and let the system learn from future queries after the update. Second, the system can predict the expected utility of each rule, and choose to update or re-learn a subset of invalid rules. Third, the system can update or re-learn all rules after the update. We plan to experiment with all of these alternatives and propose an approach to let the system decide which update alternative is the most appropriate for an expected model of database change.

Bibliography

[1] James F. Allen, Henry A. Kautz, Richard N. Pelavin, and Josh D. Tenenberg. *Reasoning About Plans.* Morgan Kaufmann, San Mateo, 1991.

[2] Hussein Almuallim and Tom G. Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence(AAAI-91)*, pages 547–552, Anaheim, CA, 1991.

[3] Yigal Arens. Services and information management for decision support. In *AISIG-90: Proceedings of the Annual AI Systems in Government Conference*, George Washington University, Washington, DC, 1990.

[4] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[5] Yigal Arens and Craig A. Knoblock. Planning and reformulating queries for semantically-modeled multidabase systems. In *Proceedings of the First International Conference on Information and Knowledge Management*, pages 92–101, Baltimore, MD, 1992.

[6] Christer Backstrom. Finding least constrained plans and optimal parallel executions is harder than we thought. In *Current Trends in AI Planning: EWSP'93–2nd European Workshop on Planning*, Frontiers in AI and Applications. IOS Press, Amsterdam, 1993.

[7] Anthony Barrett, Keith Golden, Scott Penberthy, and Daniel Weld. UCPOP user's manual (version 2.0). Technical Report 93-09-06, Department of Computer Science and Engineering, University of Washington, 1993.

[8] J.M. Blanco, A. Illarramendi, and A. Go ni. Using a terminological system to integrate relational databases. Facultad de Informatica, Universidad del País Vasco, Apdo 649, San Sebastián, Spain, 1992.

[9] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[10] Yandong Cai, Nick Cercone, and Jiawei Han. Learning in relational databases: An attribute-oriented approach. *Computational Intelligence*, 7(3):119–132, 1991.

[11] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In Kurt VanLehn, editor, *Architectures for Intelligence*, pages 241–278. Lawrence Erlbaum, Hillsdale, NJ, 1991.

[12] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[13] Arvola Chan, Sy Danberg, Stephen Fox, Wen-Te K. Lin, Anil Nori, and Daniel Ries. Storage and access sturctures to support a semantic data model. In *Proceedings of the 8th International Conference on Very Large Data Bases*, pages 122–130, Very Large Database Endowment, Saratoga, CA, 1982.

[14] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[15] Vasek. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4, 1979.

[16] Christine Collet, Michael N. Huhns, and Wei-Min Shen. Resource integration using a large knowledge base in Carnot. *IEEE Computer*, pages 55–62, December 1991.

[17] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction To Algorithms*. The MIT Press/McGraw-Hill Book Co., Cambridge, MA, 1989.

[18] Ken Currie and Austin Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52(1):49–86, 1991.

[19] Charles L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern matching problem. *Artificial Intelligence*, pages 17–37, 1982.

[20] David Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[21] Alexander Horz. On the relation of classical and temporal planning. In *Proceedings of the Spring Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, 1993.

[22] Chun-Nan Hsu and Craig A. Knoblock. Reformulating query plans for multidatabase systems. In *Proceedings of the Second International Conference on Information and Knowledge Management*, Washington, D.C., 1993. ACM.

[23] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[24] A. Illarramendi, J.M. Blanco, and A. Go ni. One step to integrate data and knowledge bases. Facultad de Informatica, Universidad del País Vasco, Apdo 649, San Sebastián, Spain, 1992.

[25] Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2):111–152, 1984.

[26] Subbarao Kambhampati. Multi-contributor causal structures for planning: A formalization and evaluation. *Artificial Intelligence*, Fall, 1994.

[27] Jonathan Jay King. *Query Optimization by Semantic Reasoning*. PhD thesis, Stanford University, Department of Computer Science, 1981.

[28] Craig A. Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Canada, 1994.

[29] D. Lenat and R.V. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Reading, MA, 1990.

[30] Robert MacGregor. A deductive pattern matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Saint Paul, Minnesota, 1988.

[31] Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1990.

[32] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*. Anaheim, CA, 1991.

[33] Donald P. McKay, Timothy W. Finin, and Anthony O'Hare. The intelligent database interface: Integrating AI and database systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990.

[34] Ryszard S. Michalski. A theory and methodology of inductive learning. In *Machine Learning: An Artificial Intelligence Approach*, volume I, pages 83–134. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1983.

[35] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* PhD thesis, Computer Science Department, Carnegie Mellon University, 1988.

[36] Steven Minton, John Bresina, and Mark Drummond. Commitment strategies in planning: A comparative analysis. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[37] Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel R. Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40(1-3):63–118, 1989.

[38] Marlon Nunez. The use of background knowledge in decision tree induction. *Machine Learning*, 6:231–250, 1991.

[39] Mike P. Papazoglou, Steven C. Laufmann, and Timos K. Sellis. An organizational framework for cooperating intelligent information systems. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):169–202, 1992.

[40] Micheal J. Pazzani and Dennis Kibler. The utility of knowledge in inductive learning. *Machine Learning*, 9:57–94, 1992.

[41] J. Scott Penberthy. *Planning with Continuous Change.* PhD thesis, Department of Computer Science and Engineering, University Washington, 1993.

[42] J. Scott Penberthy and Daniel S. Weld. UCPOP: A sound, complete, partial order planner for ADL. In *Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 189–197, Cambridge, MA, 1992.

[43] Gregory Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro, editor, *Knowledge Discovery in Databases*, pages 229–248. MIT Press, 1991.

[44] M.P. Reddy, B.E. Prasad, and P.G. Reddy. Query processing in heterogeneous distributed database management systems. In Amar Gupta, editor, *Integration of Information Systems: Bridging Heterogeneous Databases*, pages 264–277. IEEE Press, NY, 1989.

[45] Pierre Regnier and Bernard Fade. Complete determination of parallel actions and temporal optimization in linear plans of action. In J. Hertzberg, editor, *European Workshop on Planning*, pages 100–111. Springer-Verlag, 1991.

[46] Stuart J. Russell. *The Use of Knowledge in Analogy and Induction.* Morgan Kaufmann, San Mateo, CA, 1989.

[47] Shashi Shekhar, Babak Hamidzadeh, Ashim Kohli, and Mark Coyle. Learning transformation rules for semantic query optimization: A data-driven approach. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):950–964, 1993.

[48] Shashi Shekhar, Jaideep Srivastava, and Soumitra Dutta. A formal model of trade-off between optimization and execution costs in semantic query optimization. In *Proceedings of the 14th VLDB Conference*, Los Angeles, CA, 1988.

[49] Wei-Min Shen. Discovering regularities from knowledge bases. *International Journal of Intelligent Systems*, 7:623–635, 1992.

[50] Sreekumar T. Shenoy and Zehra Meral Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):344–361, 1989.

[51] Michael D. Siegel. Automatic rule derivation for semantic query optimization. In Larry Kerschberg, editor, *Proceedings of the Second International Conference on Expert Database Systems*, pages 371–385. George Mason Foundation, Fairfax, VA, 1988.

[52] Ming Tan. Cost-sensitive learning of classification knowledge and its application in robotics. *Machine Learning*, 13:7–33, 1993.

[53] Austin Tate. Project planning using a hierarchic non-linear planner. Research Report 25, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1976.

[54] Garold S. Tjaden and Michael J. Flynn. Detection and parallel execution of independent instructions. *IEEE Transactions on Computers*, C-19(10):889–895, 1970.

[55] Shalom Tsur and Carlo Zaniolo. An implementation of gem – supporting a semantic data model on a relational back-end. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 286–295, ACM, New York, 1984.

[56] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems*, volume I. Computer Science Press, Palo Alto, CA, 1988.

[57] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems*, volume II. Computer Science Press, Palo Alto, CA, 1988.

[58] Manuela M. Veloso. Nonlinear problem solving using intelligent casual-commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[59] Manuela M. Veloso, M. Alicia Perez, and Jaime G. Carbonell. Nonlinear planning with parallel resource allocation. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 207–212, San Diego, CA, 1990.

[60] Steven A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246–267, May 1983.

[61] Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In Daniel Weld and Johann de Kleer, editors, *Readings in Qualitative Reasoning about Physical Systems*, pages 373–381, San Mateo, CA, 1989. Morgan Kaufmann.

[62] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4), 1994.

[63] David E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22(3):269–301, 1984.

The SIMS Manual
Version 1.0*

José-Luis Ambite
Yigal Arens
Naveen Ashish
Chin Y. Chee
Chun-Nan Hsu
Craig A. Knoblock
Wei-Min Shen
Sheila Tejada
Information Sciences Institute
University of Southern California
4676 Admiralty Way,
Marina del Rey, CA 90292, U.S.A.

July 13, 1995

Abstract

SIMS provides intelligent access to heterogeneous, distributed information sources, while insulating human users and application programs from the need to be aware of the location of the sources, their query languages, organization, size, etc.

This manual explains how to bring up a SIMS information server in a new application domain. After providing a short overview of relevant features of the SIMS system, it describes the modeling and programming work that has to be performed to support the extension of SIMS to a given collection of information sources in the domain. To aid a user inexperienced with the technological infrastructure underlying SIMS, the manual contains examples structured as a tutorial that can be followed to actually produce a working SIMS system.

For general discussion, information and announcements concerning SIMS, please send mail to *sims-forum-request@isi.edu* and ask to be added to the **SIMS-Forum** mailing list. For bug reports. please send mail to *sims-bug-report@isi.edu*.

*The research reported here was supported in part by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under Contract Number F30602-94-C-0210, in part by a Technology Reinvestment Program award funded by the Advanced Research Projects Agency under Contract Number MDA972-94-2-0010 and by the State of California under Contract Number C94-0031, in part by the National Science Foundation under Grant Number IRI-9313993, and in part by an Augmentation Award for Science and Engineering Research Training funded by the Advanced Research Projects Agency under Contract Number F49620-93-1-0594. The views and conclusions contained in this paper are those of the author and should not be interpreted as representing the official opinion or policy of RL, ARPA, CA, NSF, the U.S. Government, or any person or agency connected with them.

Contents

1 Introduction

The overall goal of the SIMS project is to provide intelligent access to heterogeneous, distributed information sources (databases, knowledge bases, flat files, programs, etc.), while insulating human users and application programs from the need to be aware of the location of the sources. their query languages, organization. size, etc.

The standard approach to this problem has been to construct a global schema that relates all the information in the different sources and to have the user pose queries against this global schema or various views of it. The problem with this approach is that integrating the schemas is typically very difficult. and any changes to existing data sources or the addition of new ones requires a substantial, if not complete, repetition of the schema integration process. In addition, this standard approach is not suitable for including information sources that are not databases.

SIMS provides an alternative approach. A model of the application domain is created, using a knowledge representation system to establish a fixed vocabulary describing objects in the domain, their attributes and relationships among them. For each information source a model is constructed that indicates the data-model used, query language, network location, size estimates, etc., and describes the contents of its fields in relation to the domain model. SIMS' models of different information sources are independent, greatly easing the process of extending the system.

Queries to SIMS are written in the high-level uniform language of the domain model, a language independent of the specifics of the information sources. Queries need not contain information describing which sources are relevant to finding their answers or where they are located. Queries do not need to state how information obtained from different sources should be joined or otherwise combined or manipulated.

SIMS uses a planning system to determine how to retrieve and integrate the data necessary to answer a query. The planner first selects information sources to be used in answering a query. It then orders sub-queries to the appropriate information sources, selects the location for processing intermediate data, determines which sub-queries can be executed in parallel, and then initiates execution.

Changes to information sources are handled by changing models only. The changes will be considered by the planner in producing future plans that utilize information from the modified sources. This greatly facilitates extensibility.

The rest of this section presents an overview of SIMS and its architecture. In Section 2 we show the format of the queries that a user would input to SIMS and the output that should be expected. Then. we consider in more detail the specification of domain models, in Section 3, and information sources models, in Section 4. Section 5 gives a brief introduction on how to construct a wrapper for a new information source and how to communicate with the wrapper. Section 6 explains how to run SIMS both through its graphical user interface and its functional interface. Section 7 describes how to test and debug a new SIMS application. Section 8 presents the installation and system requirements. Finally, in Section 9 we show the code that would implement the example that is discussed throughout the manual. Section 10 contains a reading list of relevant papers.

1.1 Architecture and Background

A visual representation of the components of SIMS is provided in Figure 1.

SIMS addresses the problems that arise when one tries to provide a user familiar only with the general domain with access to a system composed of numerous separate data- and knowledge-bases.

Specifically, SIMS does the following:

- **Modeling:** It provides a uniform way to describe information sources to the system, so that data in them is accessible.
- **Information Source Selection:** Given a query, it
  - Determines which information sources contain the data relevant to answering the query.
  - For those concepts mentioned in the query which appear to have no matching information source, it determines if any knowledge encoded in the domain model (such as relationship to other con-

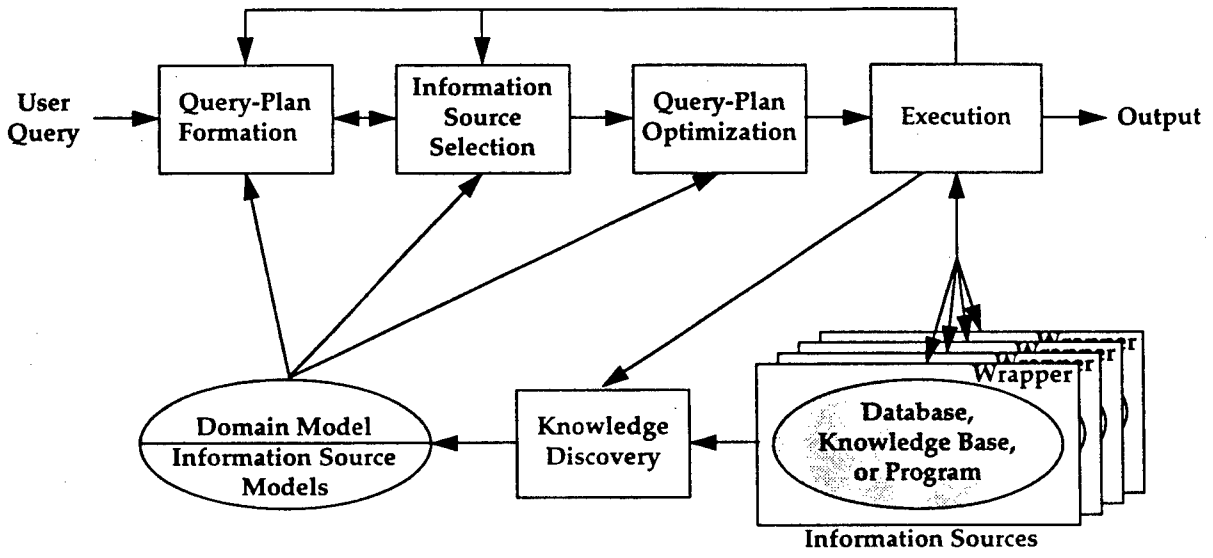


Figure 1: SIMS Overview Diagram.

cepts) permits reformulation of the query in a way that will enable suitable information sources to be identified.

- **Access Planning:** It creates a plan, a sequence of subqueries and other forms of data-manipulation that when executed will yield the desired information.
- **Query-Plan Optimization:** It uses knowledge about databases — their sizes, semantic rules concerning their contents, indexes — to optimize the plan.
- **Discovering Database Semantic Rules:** By querying databases and other information sources and analyzing the returned information, it discovers semantic rules characterizing their contents. This learned knowledge is used to modify SIMS' domain and information source models and ultimately to improve query plans. This module has not been released yet.
- **Execution:** It executes the reformulated query plan; establishing network connections with the appropriate information sources, transmitting queries to them and obtaining the results for further processing. During the execution process SIMS may detect that certain information sources are not available, or respond erroneously. In such cases, the relevant portion of the query plan will be replanned.

Each information source is accessed through a *wrapper*, a module that can translate queries to that information source from the SIMS query language to the query language understood by that source. The wrapper then takes the data returned by the information source and sends it on to SIMS in the form SIMS expects.

1.2 Information Sources Supported

In order for SIMS to support an information source there must be an information source model for it, and there must exist a wrapper for that type of source. While each information source needs to be modeled individually, only one wrapper is required for any type of information source. For example, LIM (see below) serves as the wrapper for any Oracle database.

Wrappers for Loom knowledge bases, Oracle relational databases, and MUMPS-based network databases have already been written for SIMS. To add a new database of any of these types requires, therefore, only to create an information source model for it. In order to add an information source of a new type one would

```
(sims-retrieve (?last-name)
  (:and (patient ?patient)
        (doctor-name ?patient "GONZALEZ")
        (last-name ?patient ?last-name)))
```

Figure 2: Example SIMS/Loom Query

have to obtain, or write, a new wrapper for it. In the case of another database using SQL as its query language, this would be only a simple modification of LIM.

1.3 Technological Infrastructure

This subsection is provided for readers who may not be familiar with the systems underlying SIMS. A general understanding of Loom, LIM and KQML is assumed in the rest of this manual. A list of relevant papers is provided in Section 10.

1.3.1 Loom

Loom serves as the knowledge representation system SIMS uses to describe the domain model and the contents of the information sources. In addition, Loom is used to define a knowledge base that itself serves as an information source. Loom provides both a language and an environment for constructing intelligent applications. It combines features of both frame-based and semantic network languages, and provides some reasoning facilities. As a knowledge representation language it is a descendant of the KL-ONE [1] system.

The heart of Loom is a powerful knowledge representation system, which is used to provide deductive support for the declarative portion of the Loom language. Declarative knowledge in Loom consists of definitions, rules, facts, and default rules. A deductive engine called a *classifier* utilizes forward-chaining, semantic unification and object-oriented truth maintenance technologies in order to compile the declarative knowledge into a network designed to efficiently support on-line deductive query processing. For a more detailed description of Loom, see [3, 4].

To illustrate both Loom and the form of SIMS' queries, consider Figure 2, which contains a simple semantic query to SIMS. This query requests the last names of the patients of a doctor by the name of Gonzalez. The three subclauses of the query specify, respectively, that the variable **?patient** describes a member of the model concept **patient**, that the relation **doctor-name** holds between the values of **?patient** and the string **GONZALEZ**, and that the relation **last-name** holds between the values of **?patient** and the respective values of the variable **?last-name**.

As we will see later in Figure 4, **patient** is a subconcept of **person** therefore, by inheritance, it too has **person**'s attributes, among them **last-name**.

The query specifies that the value of the variable **?last-name** be returned. A query to SIMS need not necessarily correspond to a single database query, since there may not exist one database that contains all the information requested.

1.3.2 LIM

The Loom Interface Module (LIM) [5] has been developed by researchers at Unisys to mediate between Loom and databases. It currently acts as a SIMS wrapper to relational Oracle databases, using the SQL language. LIM reads an external database's schema and uses it to build a Loom representation of the database. The Loom user can then treat concepts whose instances are stored in a database as though they contained "real" Loom instances. Given a Loom query for information about instances of that concept, LIM automatically generates an SQL query to the database that contains the information, and returns the results as though they were Loom instances. LIM focuses primarily on the issues involved in mapping a SIMS query to a single database. After SIMS has planned a query and formed subqueries, each grounded in a single database, it hands the subqueries to LIM for the actual data retrieval. SIMS handles direct queries to the Loom knowledge base on its own.

1.3.3 KQML

To simplify and modularize its interaction with databases, SIMS is designed as an intelligent agent that communicates with other information agents, which can be simple information sources or other SIMS agents. The former are regular databases which are, however, enclosed in software *wrappers* — systems that accept queries to the database in the Loom language and translate them into queries that the local DBMS can handle.

The Knowledge Query Manipulation Language (KQML) [2] is an agent communcation language that supports such messages. We have adopted KQML for our agent-to-agent communication. KQML has been designed to support knowledge-based communication. Using it, agents can transmit structured objects along with the contents of the objects, so they can transmit model fragments together with queries using terms from those models. The KQML language handles the interface protocols for transmitting queries, returning the appropriate information, and building the necessary structures.

2 The SIMS Language

The SIMS language supports three different types of commands for retrieving, modifying, and monitoring information. The command **sims-retrieve** is used for querying, while the commands **sims-insert**, **sims-delete**, and **sims-update** are transaction commands. The commands **sims-begin-notify** and **sims-end-notify** handle the monitoring of data items. In the following sections each type of command will be discussed in further detail.

2.1 Sims-Retrieve Query Command

SIMS takes a **sims-retrieve** query as input and outputs the data satisfying the constraints specified in the query. Currently, SIMS supports most features of the Loom query language. These features should be enough for most database applications. From now on, we call this subset of the Loom query language as the SIMS query language. The output format of SIMS is a (LISP) list which can contain constants, Loom objects or tuples.

```
<query> ::= (sims-retrieve ({<variable>}+) <query-expr>)
<query-expr> ::=
     ({:and | :or} {<query-expr>}+) |
     (:not <query-expr>) |
     (:collect ({<variable>}+) <query-expr>) |
     (:implies <query-expr> <query-expr>) |
     ({:for-some | :for-all} ({<variable>}+) <query-expr>) | <clause>
<clause> ::=
     <concept-exp> | <relation-exp> | <assignment-exp> | <comparison-exp> |
     <aggregation-exp>
<concept-exp> ::= (<concept-name> <variable>)
<relation-exp> ::=
    (<relation-name> {<bound-variable> | <function-exp>} {<term> | <function-exp>})
<assignment-exp> ::= (= <unbound-variable> {<arith-exp> | <set-exp>})
<comparison-exp> ::= <member-comparison> | <arithematic-comparison>
<function-exp> ::= (<relation-name> {<bound-variable> |<constant> |<symbol> |<function-exp>}) |
     (<aggregation-function> <set-exp>)
<set-exp> ::= ({<constant>}+) |(:collect ({<variable>}+)<query-expr>) |<function-exp> |
     <bound-variable>
<aggregation-exp> ::=
     (<aggregation-function> <set-exp> {<clause> |<variable>})
<member-comparison> ::=
     (member <bound-variable> <set-exp>)) | (members <set-exp> <bound-variable>)
<arithematic-comparison> ::=
     (<comparison-op> {<arith-exp> | <function-exp>} {<arith-exp> | <function-exp>})
<arith-exp> ::= <number> | <bound-variable> |
     (<arith-op> {<arith-exp> | <function-exp>} {<arith-exp> | <function-exp>})
<arith-op> ::= + | - | * | /
<comparison-op> ::= = | > | < | >= | <= | !=
<aggregation-function> ::= count | avg | sum | min | max
<concept-name> ::= <symbol>
<relation-name> ::= <symbol>
<term> ::= <constant> | <variable>
<variable> ::= <bound-variable> | <unbound-variable>
<bound-variable>[1] ::= ?<symbol>
<unbound-variable> ::= ?<symbol>
<constant> ::= <number> | <string>
```

Figure 3: BNF for the SIMS Query Language

```
(<relation-name> <term>)
(<relation-name> <symbol>)
(<relation-name> <function-exp>)
(<aggregation-function> <set-exp>)
```

Some examples of function expressions are:

```
(last-name ?patient)
(last-name (used-by-patient ?room))
```

The function `last-name` returns the the last name of the patient. In the second example the function `used-by-patient` returns the patient that is in `?room` and the function `last-name` returns the the last name of that patient.

The following examples are relation expressions which contain these function expressions:

```
(doctor-name ?patient (last-name ?patient))
(last-name (used-by-patient ?room) "Lee")
```

The first expression is satisfied only when the patient and the doctor have the same last name. The second expression is satisfied when the last name of the patient in the `?room` is "Lee".

- Assignment expressions:

```
(= <unbound-variable> <arith-expr>)
(= <unbound-variable> <set-exp>)
```

The first clause assigns to the unbound variable the computed result of <arith-expr>. In the second clause the variable is assigned to a <set-exp> which is defined as a set of constants or any expression which returns a set of constants.

For the following example, suppose we have a concept `patient` and relations on patients and their names (`last-name` and `first-name`). The role `medi-care-fee` states the relation between patients and their medical care charge. The role `insurance-deduct` defines the insurance deduction of a patient. The following query will return a list of patients' names and total balances:

```
(sims-retrieve (?lname ?fname ?total)
          (:and (patient ?patient)
                (last-name ?patient ?lname)
                (first-name ?patient ?fname)
                (medi-care-fee ?patient ?mfee)
                (insurance-deduct ?patient ?insd)
                (= ?total (- ?mfee ?insd))))
  ==> (("Okumura" "Ben" 15000)
       ("DeSpain" "Ann" 20000)
       ("Kumar" "Barbara" 18500)
       ("Hamilton" "Sheila" 27500)
       ("Lee" "Kayano" 26500)
       ⋮
```

- Comparison expressions are used to express a constraint on variables. The following are forms of member comparisons:

```
(member <bound-variable> <set-exp>)
(members <set-exp> <bound-variable>)
```

where a <set-exp> is defined as a set of constants or any expression which returns a set of constants. The first clause is satisfied if the variable is bound to one of the constants (i.e., strings or numbers) in the <set-exp>. In the second clause the order of the arguments are reversed.

The following are examples of member comparisons:

```
(member ?lname ("Smith" "Hamilton" "DeSpain" "Datta"))
(members (doctor-name ?patient) ?doctor)
```

The BNF syntax for the SIMS query language is shown in Figure 3. We next provide a more in-depth description of the language. The following is the basic form of a SIMS query:

```
(sims-retrieve (?v1 ... ?vn) <query-expr>)
```

The variables listed after the **sims-retrieve** command, $?v_1 \ldots ?v_n$, are considered output variables. This means that the values of these variables are returned as the output of the query. All variables must be named with the prefix '?'. The query expression is composed of clauses and constructors. Clauses determine the values of the variables by binding the variables to specific types of values. In other words, clauses constrain the values of the variables. There are five types of clauses supported by the SIMS language which will be described in the next section. Clauses can be grouped by constructors into queries. Currently, the set of constructors provided is **:and**, **:or**, **:not**, **:for-all**, **:for-some**, and **:collect**.

A SIMS query returns as output a list of instantiations of the output variables which satisfy the bindings of the clauses in the query body. The following shows an example of output from a SIMS query.

```
(sims-retrieve (?patient ?lname)  (:and (Patient ?patient)
                                        (last-name ?patient ?lname)))
==> ((|i|PATIENTS145443 "Richardson")
     (|i|PATIENTS145437 "Brown")
     (|i|PATIENTS145441 "Kumar") ...)
```

In this query the output variable **?patient** is bound to the concept **Patient** and the variable **?lname** is bound to the values of the role **last-name**, respectively. SIMS returns a set of tuples which contains Loom objects and strings corresponding to the instances of **Patient** and **last-name**. Loom objects are displayed with the prefix **|i|**, which is a Loom internal identification symbol.

2.1.1 Clauses

Clauses are expression that constrain the values which can be bound to a variable. A clause is satisfied when there exists values which satisfy the constraints on the variables in that clause. The following are the five types of clauses:

- Concept expressions:

  ```
  (<concept-name> <variable>)
  ```

  where <concept-name> is the name of a concept, the variable is bound to an instance of the concept <concept-name>. An example of a concept expression is:

  ```
  (Patient ?patient)
  ```

  This constrains the variable **?patient** to only the instances of the concept **Patient**.

- User-defined relation expressions:

  ```
  (<relation-name> <bound-variable> <term>)
  (<relation-name> <bound-variable> <function-exp>)
  (<relation-name> <function-exp> <term>)
  (<relation-name> <function-exp> <function-exp>)
  ```

  where <relation-name> is the name of a relation, <bound-variable> is a variable while <term> can be either a variable or a constant (a number or a string). The first clause states that there is a binary relation <relation-name> between <bound-variable> and <term>. The following are examples of this type of relation expression:

  ```
  (last-name ?patient ?lname)
  (first-name ?patient "Ann")
  ```

  The first expression is only satisfied if the value for **?lname** is the last name of **?patient**. The second expression is only satisfied if "Ann" is the first name of **?patient**.

  The other types of relation clauses contain function expressions, <function-exp>. A function expression is basically the same as a relation expression, except that the second argument of the relation is returned as the result. The expression <function-exp> returns a constant, in this case. A <function-exp> is defined as either of the following:

[1] A variable is considered to be bound if it appears earlier in the query

The first expression is only satisfied if the value for `?lname` matches one of the four strings in the set. The second expression is only satisfied if the value for `?doctor` matches one of the strings in the set returned by the function `doctor-name`. The function `doctor-name` returns the names of the doctors for `?patient`.

Another type of comparison expression uses the arithmetic comparison operators: `=`, `>`, `<`, `>=`, `<=`, `!=`. In this case, the expression <function-exp> returns a constant.

```
(<comparison-op> <arith-expr> <arith-expr>)
(<comparison-op> <arith-expr> <function-exp>)
(<comparison-op> <function-exp> <arith-expr>)
(<comparison-op> <function-exp> <function-exp>)
```

The following are examples of the arithmetic comparison:

```
(!= (last-name ?patient) "Kumar")
(= (insurance-deduct ?patient) 300)
(> (medi-care-fee ?patient) (insurance-deduct ?patient))
```

The first example checks that the last name of the patient is not "Kumar". The = operator in the second expression tests whether the insurance deduction of the patient is equal to 300. The last example compares the medi-care fee of the patient to the insurance deduction.

- Aggregate expression:

```
(<aggregate-function> <set-exp> <term>)
```

A set of values is required as the first argument for an aggregate function. The value of the operation performed on the set of values is then bounded to <term>, as shown in the following example:

```
(count (doctor-name ?patient) ?count)
```

This counts the set of values returned by the function `doctor-name` and binds the result to the variable `?count`.

2.1.2 Query Expression Constructors

This section provides detailed descriptions for each of the expression constructors supported by SIMS. The examples refer to the models defined in later sections.

`(:and` $expr_1$ ...$expr_n$`)` — CONJUNCTION

This returns the values for which each of the expressions $expr_j$ is satisfied.

Example: `(:and (Office ?x) (hospital-room ?x))`
This expression is satisfied if `?x` is both an `Office` and a `hospital-room`.

`(:or` $expr_1$ ...$expr_n$`)` — DISJUNCTION

This returns the values for which at least one of the expressions $expr_j$ is satisfied.

Example: `(:or (Doctor ?x) (Patient ?x))`
This expression is satisfied if `?x` is either a `Doctor` or a `Patient`.

`(:for-some (?`v_1 ...`?`v_n`)` *expr*`)` — EXISTENTIAL QUANTIFICATION

This returns the values for which there exist values for the variables $?v_1$ through $?v_n$ that satisfy the expression *expr*.

Example:
```
(:for-some (?d1 ?d2)
    (:and (patient-of ?patient ?d1) (patient-of ?patient ?d2)
          (!= (doctor-id ?d1)
              (doctor-id ?d2))))
```
This expression is satisfied if there exist a `patient` which has more than one doctor.

`(:for-all (?`v_1 ...`?`v_n`) (:implies` $expr_1$ $expr_2$`))` — UNIVERSAL QUANTIFICATION

This returns the values of all sets of bindings of the variables $?v_1$ through $?v_n$ that satisfy the expression $expr_1$ and the expression $expr_2$.

Example:
```
(:for-all (?doctor)
    (:implies (patient-of ?patient ?doctor)
        (:not (doctor-Id ?doctor 135))))
```
This expressions is satisfied if all of the doctors of **?patient** do not have the identification number 135.

All of the universally quantified variables $?v_j$ must appear within $expr_1$. This is necessary in order to bind the variables to specific types of values. The bindings for the expression $expr_2$ can then be generated, because the types of the variables have already been determined.

`(:not` *expr*`)` — NEGATION

This returns the values for which expression *expr* can not be proved satisfiable.

Example:
```
(:and (Patient ?p) (:not (elderly-patient ?p)))
```
This expression is satisfied if **?p** is a **Patient** and **?p** is not known to be an **elderly-patient**.

In the negated expression *expr* variables must be bound when *expr* is evaluated. The following query is not legal because the variable $?p$ is not bound yet.
```
(retrieve ?p (:not (Patient ?p)))
```

`(:collect` *?v expr*`)` — COLLECT SATISFYING VALUES (COMPUTED SET)

This returns the list of values bound to the variable $?v$ such that *expr* is satisfied.

Example:
```
(:collect ?id
    (:for-some ?patient
        (:and (Patient ?patient)
              (patient-id ?patient ?id))))
```
Returns the list of identification numbers of all the patients.

2.2 SIMS Transaction Commands

SIMS supports transactions of insert, delete, and update under the following assumptions:

- Every transaction must have roles that can be used to uniquely identify one and only one instance of a domain concept.
- Can only add/update/delete one instance per transaction.
- The current release assumes all sub-transactions generated by SIMS can be executed successfully. In the next release, we shall have the complete two-phase commit in place.

2.2.1 Sims-Insert, Sims-Update, and Sims-Delete

- **sims-insert**: for creating a new instance of a concept with given roles/values. This function returns T for success, NIL for a failure.
- **sims-delete**: for deleting an existing instance of a concept. This function returns T for success. NIL for a failure.
- **sims-update**: for changing values of roles for an existing instance of a concept. This function returns T for success, NIL for a failure.

2.2.2 Examples

Assume that we have a domain concept called **patient**, and it has three roles: **patient-id** (key), **sex**, and **religion**. Furthermore, assume that there are two information sources: DB1 and DB2. DB1 contains patient's id and sex, and DB2 contains patient's id and religion. Then the following is a trace of transactions illustrate how to use sims-insert, sims-delete, and sims-update.

Verify there is no patient with ID 111

```
(sims-retrieve (?p)
   (:and (patient ?p)
         (patient-id ?p 111)))  ⟹ NIL
```

Create a new patient with ID 111. Note that this will trigger creations of new instances in both DB1 and DB2.

```
(sims-insert ()
   (:and (patient ?p)
         (sex ?p "M")
         (religion ?p "WHO KNOWS")
         (patient-id ?p 111)))      ⟹  T
```

Query this new patient

```
(sims-retrieve (?p ?s)
   (:and (patient ?p)
         (patient-id ?p 111)
         (sex ?p ?s)
         (religion ?p ?r)))   ⟹  (("M"  "WHO KNOWS"))
```

Change values of this patient

```
(sims-update ()
   (:and (patient ?p)
         (patient-id ?p 111)
         (sex ?p ?sex)
         (= ?sex "F")
         (religion ?p ?religion)
         (= ?religion "EVERY ONE KNOWS"))) ⟹ T
```

Query for the new values

```
(sims-retrieve (?p ?s)
   (:and (patient ?p)
         (patient-id ?p 111)
         (sex ?p ?s)
         (religion ?p ?r)))   ⟹  (("F"  "EVERY ONE KNOWS"))
```

Delete a patient with ID 111

```
(sims-delete ()
   (:and (patient ?p)
         (patient-id ?p 111)))  ⟹ T
```

2.3 SIMS Active Notifications

Clients can ask SIMS to monitor certain data items with specified conditions and actions. SIMS will execute these actions and send a notification via TCP/IP to the client whenever the data items experience changes that satisfy the specified conditions.

2.3.1 Sims-Begin-Notify and Sims-End-Notify

sims-begin-notify

```
(sims-begin-notify
   :concept 'CNAME          ;; The name of the concept to be monitored
   :when    Q1              ;; A SIMS query used as the trigger condition
   :do      #'FUN           ;; Any lisp function that takes two arguments;
```

```
                          ;; the first argument is bound to the concept name,
                          ;; and the second the transaction itself.
   :host    Address       ;; a string for the LISTENER's machine address
   :port    2020)         ;; a port number of the LISTENER

   ===> NOTIFICATION-ID ;; the function returns a notification id.
```

sims-end-notify

```
(sims-end-notify NOTIFICATION-ID)  ;; this function informs SIMS to stop
                                   ;; monitor activities specified in the
                                   ;; notification with the NOTIFICATION-ID.
```

2.3.2 Examples

Assume that there is a machine xxx.isi.edu that has a port 2020 open for incoming messages. Here is how you ask SIMS to start a notification on the concept "finding" where you are interested in knowing any transactions that involve a patient who has an injury at "left front chest":

```
(sims-begin-notify
  :concept 'finding
  :when '(sims-retrieve (?f ?id)
              (:and (finding ?f)
                    (finding-location ?f ?l)
                    (= ?l "left front chest")))
  :do #'show-transaction
  :host "xxx.isi.edu"
  :port 2020)          ⟹  NOTIFICATION-12
```

where **show-transaction** is a function that prints out its second argument. Suppose now, someone else has successfully created a new instance of "finding" as follows:

```
(sims-insert ()
      (:and (finding ?f)
            (finding-id ?f 3)
            (patient-name ?f "James Wilkie")
            (finding-location ?f "left front chest")))
```

Then SIMS will send the following message to the port 2020 on xxx.isi.edu:

```
SIMS NOTIFY: (INSERT ()
               (:AND (FINDING ?F)
                     (FINDING-ID ?F 3)
                     (PATIENT-NAME ?F "James Wilkie")
                     (FINDING-LOCATION ?F "left front chest")))
```

You can stop this notification any time you send SIMS the following message:

```
(sims-end-notify 'NOTIFICATION-12)
```

You will receive a symbol "T" if the cancellation is successful.

3 The Domain Model

A domain model provides the general terminology for a particular application domain. This model is used to unify the various information sources that are available and provide the terminology for accessing those information sources. Throughout this section we use a simple example from a medical domain that involves maintaining patient records and hospital room assignment. The example is very simple in order to provide a complete, but short, description of the model for the example.

The model is described in the Loom language, which is a member of the KL-ONE family of KR systems. In Loom, objects in the world are grouped into "classes" with a set of "roles" defined on each class. See Figure 4 for a small fragment of the domain model. Classes are indicated with circles, roles with thin arrows, and subclass relations with thick arrows. Roles are inherited down to subclasses.

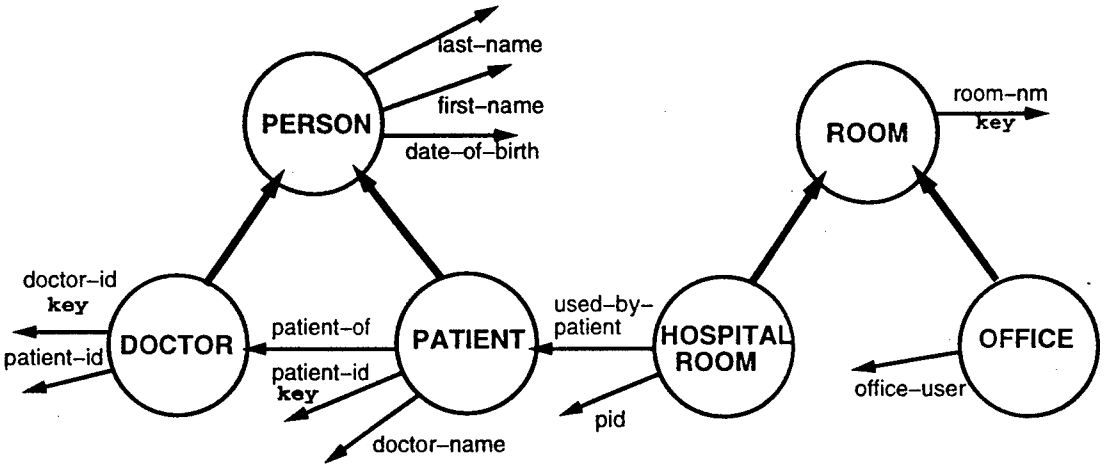


Figure 4: Domain Model Fragment

For example, there is a node in the model representing the class of `person`, a node representing the subclass of `patient`, and a `patient-of` relation specified between `patient` and `doctor`. The definition of the `patient` class is shown in Figure 5.

```
(defconcept Patient
  :is-primitive
  (:and Person
        (:all patient-id String)
        (:all patient-of Doctor)
        (:all doctor-name String))
  :annotations ((key (patient-id))))

(defrelation patient-id
  :domain Patient
  :range String)

(defrelation patient-of
  :domain Patient
  :range Doctor)

(defrelation doctor-name
  :domain Patient
  :range String)
```

Figure 5: Domain-level definition of the `Patient` class and corresponding roles

Other facts about the domain that are represented in the model include which roles on a class (if any) constitute **key roles**. These are roles that uniquely identify instances of the class that forms the domain of the relation. For example, the `patient-id` relation is a key for the class `patient`, as shown in the annotation field of the definition of patient (Figure 5). Key roles are important and powerful, because they help SIMS determine how joins can be performed. The model indicates that the patient-id uniquely identifies a patient, and thus *any* two related patient classes that both have a `patient-id` role can be joined over the patient ID (provided, of course, that they are rendered identically, such as using the same capitalization). See Section 4 for more relevant details.

The entities included in the domain model are not necessarily meant to correspond directly to objects described in any particular information source. The domain model is intended to be a description of the application domain from the point of view of someone who needs to perform real-world tasks in that domain and/or to obtain information about it.

For example, the class of elderly patients, which are patients that are 65 or older, might be particularly important for a given application, yet there may be no information source that contains only this class of patients. Nevertheless, we can define this class in terms of other classes for which information is available. The Loom definition of this class is shown in Figure 6. Notice that the definition of elderly patient requires defining another class for older-than-65, which in turn includes a simple Lisp function for computing the age of a patient.

```
(defconcept Elderly-Patient
  :is (:and Patient
    (:satisfies (?p)
                (:for-some (?dob)
                           (:and (Patient ?p)
                                 (date-of-birth ?p ?dob)
                                 (older-than-65 ?dob))))))

(defconstant *YEAR* 95)

(defconcept older-than-65 :is
  (:and Number
        (:predicate (dob)
                    (<= 65 (- *YEAR*
                              (- dob
                                 (* 100
                                    (truncate (/ dob 100)))))))))
```

Figure 6: Example of a Defined Class

When viewing model fragments such as that in Figure 4, one must remember that every fact about the domain must either be available in some information source or it must be explicitly represented. For example, consider the relation `used-by-patient`. It is tempting to believe that it stands for a mapping of hospital-rooms to the patients in the those rooms. However, all the figure itself expresses is that it is a mapping between hospital-rooms and patients and that its name is `used-by-patient`. To define the relationship, the model definition includes the following Loom statement shown in Figure 7 (which is not in the original figure because it is difficult to express graphically). It states how the `used-by-patient` relation is related to another one, `patient-id`, which relates `hospital-room` to the patients in that room. This latter relation is determined by its interpretation in some database table, as we will see later. This knowledge will, naturally, be of use in the course of processing this query.

The domain model classes are used as the basis for the **query language** that enables the user to query the information source. It is also the language in which one describes the contents of a new information source to SIMS. This is done by describing how the terms in the information source model relate to the terms in the domain model (see next section for details). In order to submit a query to SIMS, the user composes a Loom statement, using terms and roles in the domain model to describe the precise class of objects that are of interest. If the user happens to be familiar with particular information sources and their representation,

```
(defrelation used-by-patient
  :is (:satisfies (?room ?patient)
                  (:for-some (?pid)
                             (:and (Hospital-Room ?room)
                                   (Patient ?patient)
                                   (pid ?room ?pid)
                                   (patient-id ?patient ?pid)))))
```

Figure 7: Definition of the **used-by-patient** Relation

those classes and roles may be used as well. But such knowledge is not required. SIMS is designed *precisely* to allow users to query it without such specific knowledge of the data's structure and distribution.

The next section describes how information sources are described in SIMS and how their relationship to higher-level domain model classes and roles is specified.

4 Information Source Models

4.1 Modeling the Contents of an Information Source

Each information source is incorporated into SIMS by modeling the information sources and relating those models to the domain model. Appropriate classes in the domain model are linked to representations of the classes of instances contained in the database, or other information source. These mappings include the information SIMS needs to make decisions about when and whether to retrieve data from the information source in order to satisfy a user's query.

To illustrate the principles involved in representing an information source within SIMS, let us consider a table in a relational database. In SIMS, we "translate" the table into the Loom model as follows (see Figure 8).

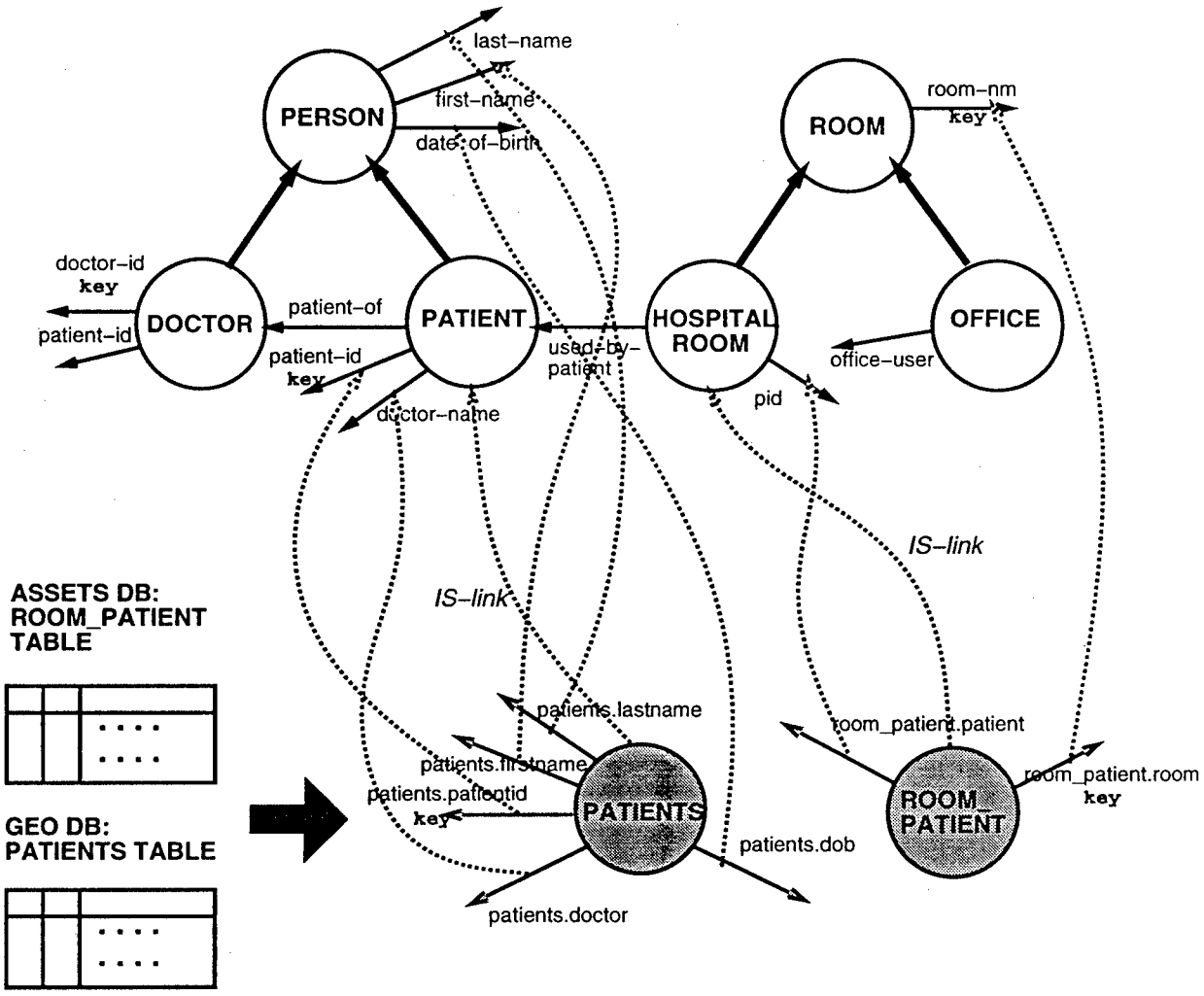


Figure 8: A Model of a Database Table Embedded in the Domain Model

The table is represented by a class that stands for the rows of the table. One may view each instance of that class as corresponding to one of the data items conceptually underlying the table in question. For example, for the `Patients` table in the `GEO` database we will create the Loom class `Patients` whose instances stand for the patients described in that table. The definition of this class is shown in Figure 9. The new class is marked as an *information source class* by specifying the "Info-Source" annotation, which indicates the source containing the corresponding data. In the figure above we have indicated that by filling in the

class in question in grey.

```
(defconcept Patients
  :is-primitive
  (:and
        (:the Patients.patientid String)
        (:the Patients.lastname String)
        (:the Patients.firstname String)
        (:the Patients.dob Number)
        (:the Patients.doctor String))
  :mixin-classes (info-source-class)
  :annotations ((Info-Source geo)))
```

Figure 9: Example Source-Level Class Definition

Each column in the table is represented in Loom as a role whose domain is the class standing for the table, and whose range is the class from which the values in the column in question are drawn. For example, the `patientid` column in the `Patients` table of the `GEO` database is represented as the Loom role `Patients.patientid`, as shown in Figure 10.

```
(defrelation Patients.patientid
  :domain Patients
  :range Number)
```

Figure 10: Example Source-Level Role Definition

Finally, each new source class must be correctly related to a class in the domain model. This is done by defining an `IS-link` between the new class and one (or more) in the domain model. An `IS-link` is the way of making explicit the semantics of the information in a given information source. In general, the name of a class is not sufficient to define the semantics of that class. A class may contain names, but the significance of those names is not self-evident. Are they indeed the names of the individual patients? Or, are they the names of the closest relative of the patient? Are they the names of the doctor? The possibilities are endless, and the schema alone is not sufficient to choose one. In order to choose to use this data at the right time, SIMS must know the precise relationship — and an IS-link to a previously defined domain model-level class establishes it. Figure 11 shows the IS-links for the Patients class. These definitions specify that Patients.patientid maps to the patient-id of the patient class, Patients.lastname maps to the last-name of the patient class, and so on.

```
(def-IS-links Patients Patient
    ((Patients.patientid patient-id)
     (Patients.lastname last-name)
     (Patients.firstname first-name)
     (Patients.dob date-of-birth)
     (Patients.doctor doctor-name)))
```

Figure 11: Example IS-links for the Patients class

To summarize, below is the complete list of tasks that need to be performed in the process of creating an information source model describing a database table:

- Create an "information source class" representing the table.
- Create "information source roles" for each column in the table.
- Create the IS-links between the new classes and roles and the domain model.

4.2 Accessing an Information Source

In addition to specifying the contents of an information source, the system also needs to know what information sources are currently available and how to access them. This section first describes the basic commands for declaring information sources and then describes the protocol for communicating with them.

To make an information source available to the system, the name, host, and access function must be declared in advance. This provides the information required for accessing an information source. The template for declaring an information source is shown in Figure 12. The <unique identifier> provides a term for referring to a specific information source. The <name> of an information source might not be unique since you may have multiple instances of the same information source running on different hosts. The <host> is the name of the machine where the information source is running. The <initialization function> and <termination function> are optional arguments that specific functions for starting and stopping information sources (if SIMS has control over them). The <transaction function> is the function for sending a command to the information source and will be passed the command that the information source is supposed to process.

```
(define-information-source <unique identifier>
      :name '<information source name>
      :host '<information source host>
      :init-fn <initialization function (optional)>
      :term-fn <termination function (optional)>
      :transaction-fn <transaction function>)
```

Figure 12: Template for Defining an Information Source

Figure 13 shows an instantiated declaration for a local loom knowledge base.

```
(define-information-source patient-kb
      :name 'geo
      :host 'kb1
      :init-fn #'(lambda ()(format t "local-geo-kb initialized"))
      :term-fn #'(lambda ()(format t "local-geo-kb terminated"))
      :transaction-fn #'(lambda (trans)
                          (info-source-op #'execute-loom-trans trans
                                          :kb "MANUAL-KB")))
```

Figure 13: Example Definition of a Loom Knowledge Base

Figure 14 shows an instantiated declaration for an relational database that is accessed through a LIM wrapper.

```
(define-information-source patient-db
      :name 'geo
      :host 'isd10.isi.edu
      :init-fn #'(lambda ()(lim-open-db :db-name 'geo))
      :term-fn #'(lambda ()(lim-close-db 'geo))
      :transaction-fn #'(lambda (trans)
                          (info-source-op #'execute-lim-trans trans
                                          :server-name "ISI-GEO-SERVER")))
```

Figure 14: Example Definition of a Oracle Database using the LIM Wrapper

5 Information-Source Wrappers and Communication

Once the SIMS planner has selected the desired sources for a user's query and devised a plan for obtaining (or updating) the required information, it must communicate with the individual information sources. In order to modularize this process and cleanly separate query planning from communication issues, SIMS requires that for each type of information source there exist a wrapper with which it will communicate. The wrapper must be capable of translating between the SIMS query language and the information source's query language, as well as between the data output format of the information source and a format appropriate for SIMS.

This section explains how wrappers are used by SIMS. The communication flow and protocols used will be described as well.

5.1 Information Source Wrappers

An information source's wrapper will receive as input a restricted form of the SIMS query language (described in Section 2). The restiction is that all concepts and roles used in the query will be drawn *only* from that information source's model. Note that at the time when such communication takes place SIMS has already determined that the query being sent to the information source can be processed in its entirety by that source alone.

The wrapper converts a SIMS query into a query in information source's query language. It submits it to the source and returns to SIMS a list of tuples corresponding to the variable parameters used in the submitted query.

To standardize the use of information source wrappers, SIMS contains a function. `info-source-op` that makes the programmatic interface more standardized and performs the actions necessary to contact a server. This function creates Loom instances when that will be required for subsequent SIMS processing. It issues the remote KQML request if the information source is a remote server. Figure 15 illustrates the relationship between SIMS, `info-source-op`, the wrappers and the information sources.

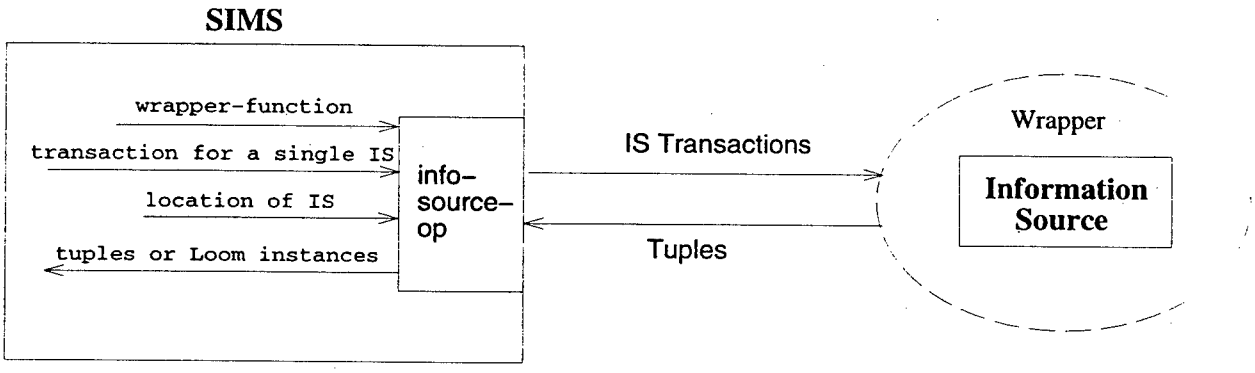


Figure 15: Communication Between SIMS and Information Sources

`info-source-op` is called with the following arguments:

```
(info-source-op wrapper-fn transaction &key server-name kb)
```

`wrapper-fn` A function that will be called with `query` as an argument, to produce an appropriate statement in the information system's query language and satisfy the transaction. This is the actual wrapper for the information source.

`query` The query. An expression of the form (`<type>` `<output arg>` `<query body>`) where `<type>` may be one of the following: `retrieve`, `insert`, `delete`, or `update`.

`server-name` If supplied, this is the name of the remote server, a string. If absent, the query will be executed on a local information source.

`kb` If supplied, the name of a knowledge base in which the query should be processed. If absent, the query will be processed in the current knowledge base.

The function will return a list of tuples which may contain Loom instances if so specified in the `<output args>` (see below). `info-source-op` should be used in the information source declaration for specifying the `transaction-fn` function (see Section 4.2).

For example, consider the simple SIMS query:

```
(sims-retrieve (?id ?fname ?lname)
               (:and (Patient ?patient)
                     (Patient-id ?patient ?id)
                     (First-name ?patient ?fname)
                     (Last-name ?patient ?lname)))
```

Assuming that this information is available from a single LIM information source, it will ultimately generate the information source query:

```
(info-source-op #'execute-lim-trans
                '(retrieve (?id ?fname ?lname)
                     (:and (Patients ?patient)
                           (Patients.patientid ?patient ?id)
                           (Patients.firstname ?patient ?fname)
                           (Patients.lastname ?patient ?lname)))
                :server-name "ISI-GEO-SERVER")
```

This is the level at which SIMS interacts with the information source server. It is the task of the information source's wrapper (in this case `#'execute-lim-trans`) to process the query beyond this point.

5.1.1 Returning Loom Instances

An added complication may arise when the SIMS query specifies that a *Loom instance* must be returned, and not simple data. Loom instances are sometimes required for further SIMS processing. Instances need to be returned when one of the output arguments in the query is a variable bound to a model concept. That is the case, for example, for the variable `?patient` in the following query:

```
(sims-retrieve (?patient ?id ?fname ?lname)
               (:and (Patient ?patient)
                     (Patient-id ?patient ?id)
                     (First-name ?patient ?fname)
                     (Last-name ?patient ?lname)))
```

As there is no need to build such a capability into each and every wrapper, we have chosen to include this functionality in `info-source-op`. This function strips off the concept variables from the output args list of the query, passes only the reformulated query to the information source wrapper, and later creates appropriate Loom instances and adds them to the data returned from the wrapper.

5.2 Remote Communication Using KQML

If an information source server is loaded into the running SIMS environment, a straight function call to the appropriate information source wrapper function is sufficient to process a query. For communication with servers running on remote hosts, SIMS uses the Knowledge Query and Manipulation Language (KQML)

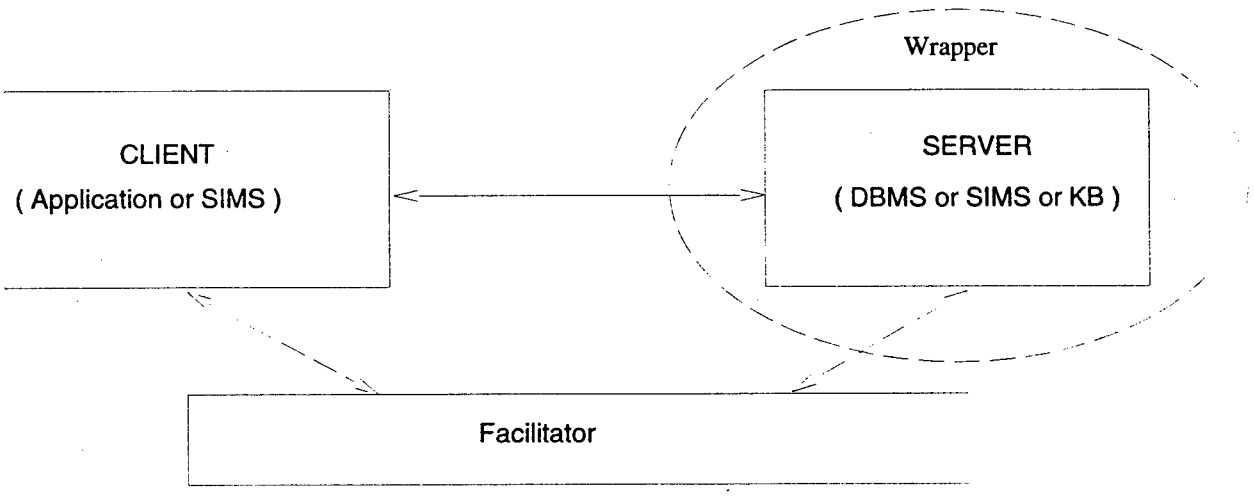


Figure 16: Communication via KQML

protocol [2]. KQML is a language for communication and knowledge sharing between autonomous programs. A simplified view of KQML-based communication is presented in Figure 16.

For our purpose, KQML provides two main types of functionality that ease the communication between clients (application programs using SIMS or SIMS itself) and servers. KQML provides a flexible standard language for client-server communication that is available for many platforms as well as implementation in different languages. It also provides a registry of all clients and servers so that a client only need to refer to the name registered on the registry by the server (which is usually the name of the service provided and hence more meaningful than just a host address) to communicate with the servers.

The central registry of services in KQML is called the ***facilitator***, and it records all KQML clients and their addresses. One can query the facilitator for services available as well as other information, we are mostly interested in the facilitator for providing the addresses of information source servers SIMS need to communicate with. (This address resolution process happens transparently and does not require user intervention.) The client and server must both be registered with the facilitator. The global variable `kqml::*local-facilitator-name*` specifies where the facilitator is located and both the client and server should agree on a facilitator accessible to both. A server registers itself by executing the command:

```
(KQML:START-KQML <server name>)
```

The `<server name>` must be unique. Clients are started by invoking `(start-kqml-client)`. This will register the client using a unique name of the following form, `<user><host>-<timestamp>`, where `timestamp` is gotten from `(get-universal-time)`. Information servers must, of course, also be declared in SIMS, as described in Section 4.2.

The way to check what is available on a facilitator, or to verify that a service that was registered is up, is to issue the command `(display-kqml-clients)` in Lisp. Or using telnet:

```
> telnet <facilitator host> 5500
...<telnet msgs>...
(ask-all :content "" :reply-with t)
...<list of services registered>...
```

Note that the KQML clients/servers only contact the facilitator once to verify the existence of a server and to get its address. The user need not know where a particular server is located but only its name (e.g., UNISYS-GEO-SERVER). KQML resolves the location (through the facilitator) transparently and caches it. The communication protocol used by KQML is TCP/IP. It creates a process that listens on a remote TCP/IP stream to detect messages from remote hosts.

The facilitator used by KQML must be accessible to both SIMS and to any users of the SIMS system

but need not be run on those systems itself. To run a facilitator at a site, execute the following command in a Unix shell:

```
kqml/C/bin/facilitator &
```

The client must know the messages supported by the server as only those can be processed. In KQML terms, SIMS acts as a mediator between the SIMS client and the information sources. A KQML mediator receives a request and either delegates it to one or more other servers, or processes it internally/locally (e.g,. in a local database). Hence the information source server needs to define a handler for the messages it will support and the client needs to know these messages and their form.

SIMS currently use only the **:ASK-ONE** KQML performative to communicate between with remote information source servers. This minimizes the number of handlers the information source KQML servers need to define. The handler will receive the following arguments:

```
(<transaction fun> <query> <kb>)
```

<query> is of the form (<type> <output args> <query body>). The handler should check that <type> is one of the supported operations (i.e., retrieve, insert, update, and delete). The <kb> argument specifies the knowledge base in which to execute the query and is optional.

Here is a simple **:ASK-ONE** handler that simply evaluates the expression received from the client:

```
(kqml::define-handler (ask-one)
   ;; content is expected to be of the form (<exec fn> <query>)
   (let* ((content (kqml::msg-content message))
          (exec-fn (car content))
          (query (cdr content))))

     (when (member (car query) '(retrieve update delete insert))
      (kqml::make-msg 'reply (apply exec-fn query))))
```

6 Running SIMS

SIMS can be run either by issuing commands to the Lisp listener or using the graphical interface. The commands that are available in both modes are described in this section.

6.1 Top-Level Commands

The top level commands of SIMS can be classified in six groups: query execution, transactions, active notification, query set management, information source management, and tracing.

6.1.1 Query Commands

The main command to execute a query is:

(sims-retrieve <parameter-list> <query-exp>)

A complete description of the query syntax is provided in Section 2. A pre-stored query (see query set management subsection) can be executed with:

(run-query <num>)

6.1.2 Transaction Commands

SIMS supports insert, delete. and update transactions as explained in Section 2.

(sims-insert <parameter-list> <instance-exp>) Creates new instances in the information sources with roles and values as given by <instance-exp>, which may be expressed in domain terms.

(sims-delete <parameter-list> <instance-exp>) Deletes the instances from the information sources specified by the (domain or source) expresion <instance-exp>.

(sims-update <parameter-list> <instance-exp>) Changes values of some roles for the source instances corresponding to the (domain or source) <instance-exp>.

6.1.3 Active Notifications

Clients can ask SIMS to monitor certain data items with specified conditions and actions. SIMS will execute these actions and send a notification via TCP/IP to the client whenever the data items experience a change to a state that satisfies the specified conditions.

(sims-begin-notify :concept <concept-name> :when <sims-query> :do <function>
:host <address> :port <port>)
When the <sims-query> is satisfied over the concept
<concept-name>, <function> is executed and the results sent to the <port> of the machine identified by <address>. This functions returns an identifier for each notification currently in the system.

(sims-end-notify <notification-id>) Informs SIMS to stop monitoring the activities specified in the notification with <notification-id>.

6.1.4 Query Set Management

Sometimes it is convenient to have frequently used queries stored in the system. A query set can be predefined by setting the global variable *queries* to the list of queries. This query set can also used from the graphical interface described in the next section.

(list-queries) Provides a list of the numbers of predefined queries.

(load-comment <num>) Retrieves the comment for query <num>.

(load-query <num>) Retrieves query <num>.

(plan-query <num>) Generates the plan for performing query <num>), but does not execute it.

(run-query <num>) Executes query <num>.

(run-queries &optional <dont-run>) Sequentially executes all queries in *queries* except the numbers in the <dont-run> list.

The syntax for the query set is:

```
(setq *queries* '(
  (<num> <comment> <query>)
  (<num> <comment> <query>)
  .
  .
  .))
```

Here is an example of a query set:

```
(setq *queries* '(
  (1 "List all patients"

     (sims-retrieve (?id ?fname ?lname ?dob ?doctor)
                    (:and (patient ?patient)
                          (patient-id ?patient ?id)
                          (first-name ?patient ?fname)
                          (last-name ?patient ?lname)
                          (date-of-birth ?patient ?dob)
                          (doctor-name ?patient ?doctor)))
     )

  (3  "Which patient is in room 101"

      (sims-retrieve (?fname ?lname)
                     (:and (patient-room ?room)
                           (room-nm ?room 101)
                           (used-by-patient ?room ?patient)
                           (first-name ?patient ?fname)
                           (last-name ?patient ?lname)))
      )
  ))
```

6.1.5 Information Source Management

The commands for manipulating the information sources are:

(list-sources) Lists all of the declared information sources.

(available-sources) Lists all of the currently available information sources that the system can access.

(initialize-source <unique-id>) Initializes the given information source.

(initialize-all-sources) Initializes all defined information sources.

(close-source <unique-id>) Closes the given information source.

(close-all-sources) Closes all of the defined information sources.

6.1.6 Tracing

In order to facilitate debugging and show the behavior of the system in a greater detail, the following commands instruct SIMS to print additional information about its processing.

(sims-trace-on) Turns on tracing. SIMS prints additional information on the query planning and execution, such as plan steps, partial reformulations, information sources accessed, intermediate results, etc.

(sims-trace-off) Turns off tracing.

(sims-trap-on) SIMS traps all errors (returning nil at the end of execution if the errors prevented the successful execution).

(sims-trap-off) When an error occurs in the processing, SIMS allows the original error handler to interrupt the execution. This command is useful when debugging an application.

6.2 The Graphical Interface

This section describes how to interact with SIMS through its graphical user interface.

The graphical interface to SIMS is invoked by calling the function **sims**, which has the optional keyword **:host**, used when the display is different from that of the machine in which SIMS is executed. Examples of invocation are: **(sims)**, or **(sims :host "sunstruck.isi.edu")** to display the interface on the machine **sunstruck.isi.edu**.

The SIMS interface (see Figure 17) is divided into three main panes: the Interaction/Trace pane (lower right quadrant), the Query pane (lower left quadrant), and the Graph pane (upper half).

The user issues commands either by selecting a command in the Command menu or typing the command in the Interaction/Trace pane. Command completion is supported. This is achieved by typing the first few keystrokes of a command and if unique, a space will complete it. Typically, an interaction sequence will proceed as follows (note that in the example below the commands can be the menu commands or typed in ones):

1. Select **Load Query** and choose a query to solve. The chosen query will be displayed in the Query pane.
2. To produce a plan to solve the query, select **Plan Query**. A graph of the generated plan will be displayed in the graph pane.
3. To perform the actual retrieval, once a plan has been generated, select **Execute Plan**. The appropriate plan graph will be shown in the Graph pane and the state of the execution is indicated by highlighting the currently executing node in the graph. The final answer will be displayed in the Interaction/Trace pane.

6.2.1 Graphical Interface Commands

Load Query Brings up a menu of the set of queries currently loaded in the system (in the variable ***queries***). This is the query that will be used by **Plan Query** and **Solve**. The selected query will be displayed in the bottom left pane.

Edit Query Once a query has been input to SIMS, we may want to issue a similar one. It is often faster to modify a loaded query than to retype one from scratch. This command allows the user to edit the currently selected query. Several checks are made to verify the consistency of the query. For example, concepts and roles must be defined in the SIMS knowledge base.

Set Current Query Allows the user to set the query to be processed by the interface by allowing the user to type it in the interaction/trace pane.

Text Edit Query Starts a text editor (emacs) to freely edit/input a query.

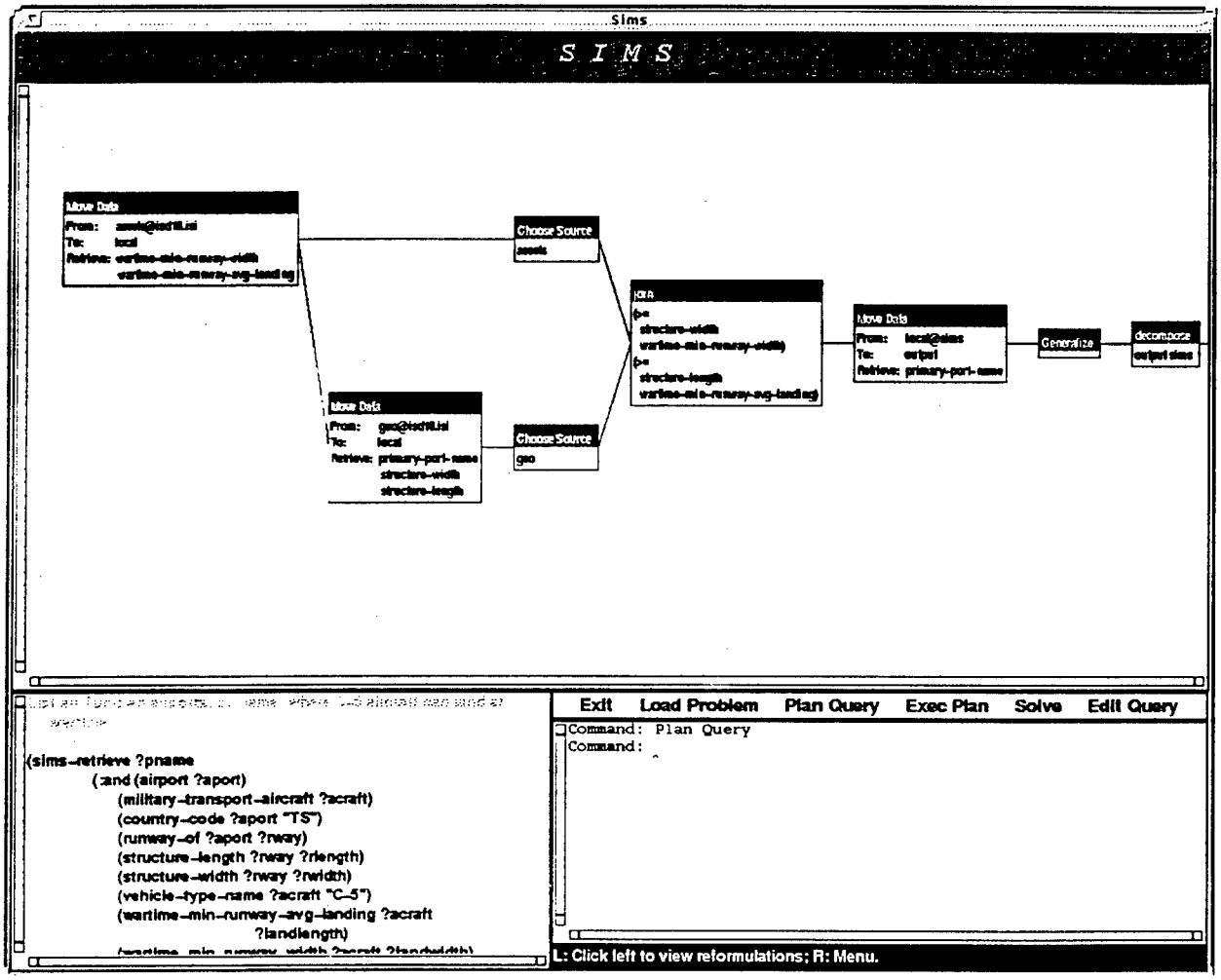


Figure 17: SIMS graphical interface

Plan Query Generates a plan for the current query. The query could have been loaded using `Load Problem`, or directly. If a plan is found, its graph will be drawn in the graph pane.

Execute Plan Executes the current plan. The node currently being executed will be highlighted (reverse video) while executed and un-highlighted when done. The final answer will be displayed in the Interaction/Trace pane.

Solve Problem Combines the previous two steps. that is, generates the plan for the currently selected query and executes it.

Exit Quits SIMS.

Display Answers Displays the results of the last executed query. The user is prompted for the number of retrieved data items to display - the default is to display all.

! <expr> Allows the evaluation of <expr>, that is, it will behave like a Lisp listener.

6.3 Plan Cost Evaluation

The SIMS architecture allows the user to change the policy of the generation of query access plans to account for different cost models. The function `set-evaluation-function` establishes the function that will guide this generation.

Currently, SIMS provides two functions. The first one, `ucpop::evaluate-plan-cost`, generates plans with the minimum number of steps. The second one, `ucpop::evaluate-plan-cost-by-size`, produces query plans in which the size of intermediate data transmitted from the information sources and processed in local joins is minimized. It uses a series of traditional database techniques to estimate the size of the queries. It considers both the expected number of tuples that a query will produce and the projection attributes. In order to calculate this estimate, it uses some statistics computed from the current contents of the information sources, such as. number of instances of a concept, number of distinct values (present in the source) of an attribute, and maximum and minimum values for numeric attributes.

Generally, `ucpop::evaluate-plan-cost-by-size` both improves the efficiency of the planning process (2 to 5-fold speed-up) and the quality of the generated plans. For complex queries this should be the function of choice. For simple queries the performance of both functions is similar. Nevertheless, size estimation needs statistics that may or may not be available for some sources. The function `gather-stats-by-info-source` will generate a set of files (one per information source) with the computed statistics for the current domain. These files can then be loaded as desired into SIMS, or incorporated into the defsystem definition of the current model to be loaded automatically.

In summary,

- to use `ucpop::evaluate-plan-cost` (the default), evaluate:

  ```
  > (set-evaluation-function #'ucpop::evaluate-plan-cost)
  ```

- to use `ucpop::evaluate-plan-cost-by-size`. evaluate:

  ```
  > (set-evaluation-function #'ucpop::evaluate-plan-cost-by-size)
  ```

- to create the statistics files, evaluate:

  ```
  > (gather-stats-by-info-source)
  ```

7 Trouble Shooting

What do you do once you have built the wrappers for your information sources, defined the domain and information source models, and submitted the first query to SIMS only to find that it does not work? We recommend that you first test your system incrementally from the bottom up by testing the wrappers to the information sources, then testing the source-level queries, and finally testing your domain-level queries. This section describes each of these in turn:

7.1 Testing the Information-Source Wrappers

Before invoking SIMS, individual wrappers for all of the information sources that are to be used should be thoroughly tested. Each wrapper should accept a source-level query as input and return a set of tuples that are the answer to that query. To test the individual wrappers, invoke the access-function for each wrapper that is defined in Section 4.2. For example the access function for a LIM Oracle database is:

```
#'(lambda (query)
      (info-source-op 'lim::execute-lim-query
                      (second query)
                      (third query)))
```

This access function can be tested directly by defining it as a function:

```
(defun lim-wrapper (query)
      (info-source-op 'lim::execute-lim-query
                      (second query)
                      (third query)))
```

Then call this function on a source-level query:

```
(lim-wrapper '(retrieve (?id ?lname ?dob)
                        (:and (patients ?patient)
                              (patients.patientid ?patient ?id)
                              (patients.lastname ?patient ?lname)
                              (patients.dob ?patient ?dob))))
(("P1001" "Okumura" 20561) ("P1002" "DeSpain" 120442)
 ("P1003" "Kumar" 63065) ("P1004" "Cooper" 101030)
 ("P1005" "Brown" 30152) ("P1006" "Smith " 71570)
 ("P1007" "Smith" 91851) ("P1008" "Chame " 12745)
 ("P1009" "Kayano" 111740) ("P1010" "Hamilton" 30359)
 ("P1011" "Hammer" 63077) ("P1012" "Dosek" 41563)
 ("P1013" "Wills" 51772) ("P1014" "Richardson" 12268)
 ("P1015" "Mizushima" 40761))
```

If this does not return the expected data, one must determine the cause of the problem and fix it before continuing to the next step.

7.2 Testing the Source-level Queries

Once all of the wrappers are working correctly, it is time to begin testing the source-level queries in SIMS. The first thing to test are exactly the same source-level queries that were used to test the individual wrappers. This will ensure that the SIMS model of the information source and the actual information source are in sync.

```
(sims-retrieve (?id ?lname ?dob)
                        (:and (patients ?patient)
                              (patients.patientid ?patient ?id)
                              (patients.lastname ?patient ?lname)
                              (patients.dob ?patient ?dob)))

UCPOP Stats: Initial terms = 3 ;   Goals = 4 ;  Success (1 steps)
      Created 11 plans, but explored only 9
      CPU time:    0.0200 sec
      Branching factor:  1.111
      Working Unifies: 23
```

```
     Bindings Added: 22
(("P1001" "Okumura" 20561) ("P1002" "DeSpain" 120442) ("P1003" "Kumar" 63065) ("P1004" "Cooper" 101030)
("P1005" "Brown" 30152) ("P1006" "Smith " 71570) ("P1007" "Smith" 91851) ("P1008" "Chame " 12745) ("P1009"
"Kayano" 111740) ("P1010" "Hamilton" 30359) ("P1011" "Hammer" 63077) ("P1012" "Dosek" 41563) ("P1013"
"Wills" 51772) ("P1014" "Richardson" 12268) ("P1015" "Mizushima" 40761))
```

If you get the message:

```
>>Error: No information sources are currently available!
```

that means that the information sources were not initialized in SIMS. Do so by using the `initialize-information-source` commands:

```
(initialize-information-source 'room-kb)
local-assets-kb initialized
NIL
49
> (initialize-information-source 'patient-kb)
local-geo-kb initialized
NIL
50
```

One should also test source-level queries in SIMS that span several information sources. After testing all of the individual source-level concepts. proceed to testing the domain-level queries.

7.3 Testing the Domain-level Queries

If all the models were set up correctly. domain-level queries should execute correctly without any problems. However, people often make mistakes in constructing the models, resulting in the system failing to produce the expected results.

Upon discovering a problem, the first thing to do is to examine the relevant portion of the domain model, source model, and IS-links to see if there are any obvious errors (e.g., missing links. misspellings, etc). Correct any obvious mistakes and try again. Note that Loom often gets confused if the same concept is defined twice, so it may be best to restart things after making changes to the model.

If a complex query does not execute correctly, break it up into smaller units and test the individual parts. That will help to pinpoint the source of the problem more quickly. For example, if the following query fails:

```
(SIMS-RETRIEVE (?FNAME ?LNAME)
               (:AND (HOSPITAL-ROOM ?ROOM)
                     (ROOM-NM ?ROOM 101)
                     (PID ?ROOM ?ID)
                     (PATIENT ?PATIENT)
                     (PATIENT-ID ?PATIENT ?ID)
                     (FIRST-NAME ?PATIENT ?FNAME)
                     (LAST-NAME ?PATIENT ?LNAME)))
```

the next thing to do is to break it into smaller queries:

```
(SIMS-RETRIEVE (?FNAME ?LNAME)
               (:AND (PATIENT ?PATIENT)
                     (FIRST-NAME ?PATIENT ?FNAME)
                     (LAST-NAME ?PATIENT ?LNAME)))
```

After identifying the simplest query that fails, go back and examine the relevant portions of the model to see if it is correct. If so, the next step is to see if the system can generate a plan for the query. This is done using the `plan-query` command:

```
(plan-query '(SIMS-RETRIEVE (?FNAME ?LNAME)
               (:AND (PATIENT ?PATIENT)
                     (FIRST-NAME ?PATIENT ?FNAME)
                     (LAST-NAME ?PATIENT ?LNAME))))

Sources: ((IS-AVAILABLE ASSETS KB2) (IS-AVAILABLE GEO KB1))

UCPOP Stats: Initial terms = 2 ;   Goals = 4 ;  Success (2 steps)
     Created 29 plans, but explored only 18
     CPU time:    0.0600 sec
```

```
      Branching factor:  1.389
      Working Unifies: 68
      Bindings Added: 65
Step 1  :
(UCPOP::MOVE GEO
             KB1
             UCPOP::OUTPUT
             (RETRIEVE (?FNAME ?LNAME)
                       (:AND (PATIENTS ?PATIENT)
                             (PATIENTS.FIRSTNAME ?PATIENT ?FNAME)
                             (PATIENTS.LASTNAME ?PATIENT ?LNAME))))
Step 2  :
(UCPOP::SELECT-SOURCE UCPOP::OUTPUT
                      UCPOP::SIMS
                      (RETRIEVE (?FNAME ?LNAME)
                                (:AND (PATIENT ?PATIENT)
                                      (FIRST-NAME ?PATIENT ?FNAME)
                                      (LAST-NAME ?PATIENT ?LNAME)))
                      (RETRIEVE (?FNAME ?LNAME)
                                (:AND (PATIENTS ?PATIENT)
                                      (PATIENTS.FIRSTNAME ?PATIENT ?FNAME)
                                      (PATIENTS.LASTNAME ?PATIENT ?LNAME)))
                      GEO) .
#plan<S=3; O=0; U=0>
```

If this fails to generate a plan, then either the required sources are not available or there is still a problem with the model. If a plan is generated, but the correct data is not returned, then tracing of the execution needs to be turned on to help pinpoint the error.

```
(sims-trace-on)
```

Now rerun the problem query and the execution trace will print out each action as it is executed and the results of the individual steps. From this trace it should be possible to figure out which step or steps are failing to return the expected data.

If a steps fails during execution, by default SIMS will simply print the error message and then exit. It traps these errors so that it can attempt to replan failed actions. However, you can turn the error trapping off to investigate further using the command:

```
(sims-trap-off)
```

Now instead of trapping the error, the system will drop into the error handler and one can proceed to debug the problem.

If all else fails, create the simplest version of the domain model, source models, and query that reproduce the problem and send them to `sims-bug-report@isi.edu`.

8 Installation and System Requirements

The SIMS system currently runs in Common Lisp with MCL 2.0 on the Mac and LUCID 4.0 on Unix. We expect to have the system running in Allegro on both the PC and Unix environments shortly. SIMS requires the following software components:

LOOM provides the underlying knowledge representation and programming support. Currently using version 2.0

KQML provides remote communication support between remote DB servers and SIMS. It can also be used to communicate between multiple SIMS servers.

CLIM provides the graphical user interface. Currently using version 1.1. This component is optional since SIMS can be run without the graphical interface.

LIM provides a wrapper for accessing relational databases. Currently using version 1.1 or 1.3. If you have your own wrapper for your databases, then this is optional.

8.1 Component Structure

Here is our current component and directory structure, which we recommend that users adopt:

```
defsys - for definitions of various components and systems

planner - for the SIMS planner based on UCPOP

operators - for reformulation operators

qsize-eval-fun - evaluation function for the planner

sims-interface - for the user interface files.

domains - for domain and information source models, and queries.

sockets - TCP/IP interface to SIMS (optional)
```

8.2 Define, Load and Compile Components

We use the CMU defsystem (this comes with LOOM) to define each subsystem. Each of the above component has its own system declaration file, e.g., lim.system, planner.system. If you want to define your own subsystem, please see the files in the **defsys** directory for examples.

One can define a component that includes many other components. For example, there is a subsystem called "BASIC-SIMS" that includes: lim, kqml, planner, operators, and interface.

Before you can use the defsystems, you will need to set two global variables. The first variable, ***sims-sys-dir***, sets the directory for the location of all of the subsystems:

```
(setq *sims-sys-dir* "/home/johndoe/sims/sys/")
```

The second variable, **make::*central-registry***, sets the location of the defsystem definitions for each of the components:

```
(setq make::*central-registry* "/home/sims/sys/defsys/")
```

One can load a system using the command `make:operate-on-system`. For example, to load "basic-sims", you do:

```
(make:operate-on-system :basic-sims :load)
```

You can substitute the keyword `:load` by `:compile` to compile the system.

```
(make:operate-on-system :basic-sims :compile)
```

You can also force the system to recompile all of the files in a component by appending the `:force` keyword:

```
(make:operate-on-system :basic-sims :compile :force t)
```

The typical sequence of loading SIMS is to load the **basic-sims** first, then load the optional components that you need, and followed by the domain model and information source models that are specific for your application.

For example, after loading in the basic-sims system, you would load the example from the manual as follows:

```
(make:operate-on-system :manual-kbs :load)
```

9 Coded Example

This section gives the code that implements the example discussed throughout the manual.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; domain-model.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Domain model for the example in the SIMS user manual
;;;
(in-package :sims)
(in-kb 'sims-kb)

;;; define concepts

(defconcept PERSON
  :is-primitive
  (:and
        (:all LAST-NAME string)
        (:all FIRST-NAME string)
        (:all DATE-OF-BIRTH number))
  :annotations ((key (last-name))))

(defconcept PATIENT
  :is-primitive
  (:and PERSON
        (:all PATIENT-ID string)
        (:all PATIENT-OF DOCTOR)
        (:all DOCTOR-NAME string))
  :annotations ((key (patient-id))))

(defconcept ELDERLY-PATIENT
  :is
  (:satisfies (?p)
      (:for-some (?dob)
        (:and (PATIENT ?p)
              (DATE-OF-BIRTH ?p ?dob)
              (older-than-65 ?dob)))))

(defconstant *YEAR* 95)

(defconcept older-than-65 :is
  (:and Number
        (:predicate (dob)
            (<= 65 (- *YEAR* (- dob (* 100 (truncate (/ dob 100)))))))))

(defconcept DOCTOR
  :is-primitive
  (:and PERSON
        (:all DOCTOR-ID string))
  :annotations ((key (doctor-id))))

(defconcept ROOM
  :is-primitive
  (:all ROOM-NM number)
  :annotations ((key (room-nm))))
```

```
(defconcept HOSPITAL-ROOM
  :is-primitive
  (:and ROOM
        (:all USED-BY-PATIENT PATIENT)
        (:all PID string))
  :annotations ((key (room-nm))))

(defconcept OFFICE
  :is-primitive
  (:and ROOM
        (:all OFFICE-USER DOCTOR))
  :annotations ((key (room-nm))))

;;; define relations

(defrelation LAST-NAME
  :domain PERSON
  :range string)

(defrelation FIRST-NAME
  :domain PERSON
  :range string)

(defrelation DATE-OF-BIRTH
  :domain PERSON
  :range number)

(defrelation PATIENT-ID
  :domain PATIENT
  :range string)

(defrelation PATIENT-OF
  :domain PATIENT
  :range DOCTOR)

(defrelation DOCTOR-NAME
  :domain PATIENT
  :range string)

(defrelation DOCTOR-ID
  :domain DOCTOR
  :range string)

(defrelation ROOM-NM
  :domain ROOM
  :range number)

(defrelation PID
  :domain HOSPITAL-ROOM
  :range string)

(defrelation USED-BY-PATIENT :is
  (:satisfies (?room ?patient)
              (:FOR-SOME (?pid)
                         (:AND (hospital-room ?room)
                               (patient ?patient)
                               (pid ?room ?pid)
                               (patient-id ?patient ?pid)))))

(defrelation OFFICE-USER
  :domain OFFICE
  :range PERSON)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; queries.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example queries
;;;
(in-package "SIMS")

(setq *queries* '(

 (1 "List all patients"

 (SIMS-RETRIEVE (?id ?FNAME ?LNAME ?dob ?doctor)
                (:AND (PATIENT ?patient)
                    (patient-id ?patient ?id)
                    (FIRST-NAME ?PATIENT ?FNAME)
                    (LAST-NAME ?PATIENT ?LNAME)
                    (date-of-birth ?patient ?dob)
                    (doctor-name ?patient ?doctor)))
 )

 (2  "Which patient is in room 101"

 (SIMS-RETRIEVE (?FNAME ?LNAME)
               (:AND (HOSPITAL-ROOM ?ROOM)
                     (ROOM-NM ?ROOM 101)
                     (PID ?ROOM ?ID)
                     (PATIENT ?PATIENT)
                     (PATIENT-ID ?PATIENT ?ID)
                     (FIRST-NAME ?PATIENT ?FNAME)
                     (LAST-NAME ?PATIENT ?LNAME)))
 )

 (3  "Which patient is in room 101"

 (SIMS-RETRIEVE (?FNAME ?LNAME)
               (:AND (HOSPITAL-ROOM ?ROOM)
                     (ROOM-NM ?ROOM 101)
                     (USED-BY-PATIENT ?ROOM ?PATIENT)
                     (FIRST-NAME ?PATIENT ?FNAME)
                     (LAST-NAME ?PATIENT ?LNAME)))
 )

(4 "List elderly patients"

(SIMS-RETRIEVE (?FNAME ?LNAME ?dob)
                (:AND (ELDERLY-PATIENT ?patient)
                    (FIRST-NAME ?PATIENT ?FNAME)
                    (LAST-NAME ?PATIENT ?LNAME)
                    (DATE-OF-BIRTH ?patient ?dob)))
)
))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; source-model.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Example source model for SIMS manual
;;;
(in-package :sims)
(in-kb 'sims-kb)

;; define concepts

(defconcept PATIENTS
  :is-primitive
  (:and
        (:the PATIENTS.patientid String)
        (:the PATIENTS.lastname String)
        (:the PATIENTS.firstname String)
        (:the PATIENTS.dob Number)
        (:the PATIENTS.doctor String))
  :mixin-classes (info-source-class)
  :annotations ((Info-Source geo)))

(defconcept ROOM_PATIENT
  :is-primitive
  (:and
        (:the ROOM_PATIENT.room Number)
        (:the ROOM_PATIENT.patient String))
  :mixin-classes (info-source-class)
  :annotations ((Info-Source assets)))

;; define relations

(defrelation PATIENTS.patientid
  :domain PATIENTS)

(defrelation PATIENTS.lastname
  :domain PATIENTS)

(defrelation PATIENTS.firstname
  :domain PATIENTS)

(defrelation PATIENTS.dob
  :domain PATIENTS)

(defrelation PATIENTS.doctor
  :domain PATIENTS)

(defrelation ROOM_PATIENT.room
  :domain ROOM_PATIENT)

(defrelation ROOM_PATIENT.patient
  :domain ROOM_PATIENT)

;; define IS-links

(def-IS-links patients patient
    ((PATIENTS.patientid PATIENT-ID)
     (PATIENTS.lastname LAST-NAME)
     (PATIENTS.firstname FIRST-NAME)
     (PATIENTS.dob DATE-OF-BIRTH)
     (PATIENTS.doctor DOCTOR-NAME)))

(def-IS-links ROOM_PATIENT HOSPITAL-ROOM
    ((ROOM_PATIENT.room ROOM-NM)
     (ROOM_PATIENT.patient PID)))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; kb-source-defs.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Definition of the Loom Knowledge Base
;;;
(in-package :sims)
(in-kb 'sims-kb)

(define-information-source patient-kb
      :name 'geo
      :host 'kb1
      :term-fn #'(lambda ()(format t "local-geo-kb terminated"))
      :init-fn #'(lambda ()(format t "local-geo-kb initialized"))
      :transaction-fn #'(lambda (trans) (info-source-op #'execute-loom-trans trans)))

(define-information-source room-kb
      :name 'assets
      :host 'kb2
      :term-fn #'(lambda ()(format t "local-assets-kb terminated"))
      :init-fn #'(lambda ()(format t "local-assets-kb initialized"))
      :transaction-fn #'(lambda (trans) (info-source-op #'execute-loom-trans trans)))


(initialize-information-source 'patient-kb)
(initialize-information-source 'room-kb)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; Knowledge-Base Data
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(tell (:about patients145439 patients
      (patients.doctor "Hotz")
      (patients.dob 30359)
      (patients.firstname "Sheila")
      (patients.lastname "Hamilton")
      (patients.patientid "P1010")))
(tell (:about patients145431 patients
      (patients.doctor "Berson")
      (patients.dob 63077)
      (patients.firstname "Janice")
      (patients.lastname "Hammer")
      (patients.patientid "P1011")))
(tell (:about patients145438 patients
      (patients.doctor "Gutierrez")
      (patients.dob 40761)
      (patients.firstname "Dana")
      (patients.lastname "Mizushima")
      (patients.patientid "P1015")))

      .
      .
      .


(tell (:about room_patient145541 room_patient
      (room_patient.patient "P1015")
      (room_patient.room 101)))
(tell (:about room_patient145543 room_patient
      (room_patient.room 107)))
(tell (:about room_patient145536 room_patient
      (room_patient.patient "P1010")
      (room_patient.room 116)))

      .
      .
      .


(tellm)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; db-source-defs.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Definition of the Oracle Databases
;;;
(in-package :sims)

(define-information-source room-db
      :name 'assets
      :host 'isd10.isi.edu
      :term-fn #'(lambda ()(lim-close-db 'assets))
      :init-fn #'(lambda ()(lim-open-db :db-name 'assets))
      :transaction-fn #'(lambda (trans)(info-source-op #'execute-lim-trans trans)))


(define-information-source patient-db
      :name 'geo
      :host 'isd10.isi.edu
      :term-fn #'(lambda ()(lim-close-db 'geo))
      :init-fn #'(lambda ()(lim-open-db :db-name 'geo))
      :transaction-fn #'(lambda (trans)(info-source-op #'execute-lim-trans trans)))

(initialize-information-source 'ROOM-DB)
(initialize-information-source 'PATIENT-DB)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; lim-source-model.lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;
;;; Definition of the databases for the LIM Wrapper
;;;
(in-package :sims)
(in-kb 'sdo-kb)

;; define concepts

(defconcept PATIENTS
  :is-primitive
  (:and Db-Concept
        (:the PATIENTS.patientid String)
        (:the PATIENTS.lastname String)
        (:the PATIENTS.firstname String)
        (:the PATIENTS.dob Number)
        (:the PATIENTS.doctor String))
  :attributes :clos-class
  :annotations ((Nulls-Ok-Cols
                 (PATIENTS.lastname
                  PATIENTS.firstname
                  PATIENTS.dob
                  PATIENTS.doctor))
                (Source-Db geo)))

(defconcept ROOM_PATIENT
  :is-primitive
  (:and Db-Concept
        (:the ROOM_PATIENT.room Number)
        (:the ROOM_PATIENT.patient String))
  :attributes :clos-class
  :annotations ((Nulls-Ok-Cols
                 (ROOM_PATIENT.room
                  ROOM_PATIENT.patient))
                (Source-Db assets)))

;; define relations

(defrelation PATIENTS.patientid
  :is-primitive DB-Relation
  :domain PATIENTS
  :annotations ((Source-Db-Table PATIENTS)
                (Source-Db-Column "patientid")
                (Source-Db-Datatype String)))

(defrelation PATIENTS.lastname
  :is-primitive DB-Relation
  :domain PATIENTS
  :annotations ((Source-Db-Table PATIENTS)
                (Source-Db-Column "lastname")
                (Source-Db-Datatype String)))

(defrelation PATIENTS.firstname
  :is-primitive DB-Relation
  :domain PATIENTS
  :annotations ((Source-Db-Table PATIENTS)
                (Source-Db-Column "firstname")
                (Source-Db-Datatype String)))

(defrelation PATIENTS.dob
  :is-primitive DB-Relation
  :domain PATIENTS
  :annotations ((Source-Db-Table PATIENTS)
                (Source-Db-Column "dob")
```

```
                (Source-Db-Datatype Number)))

(defrelation PATIENTS.doctor
  :is-primitive DB-Relation
  :domain PATIENTS
  :annotations ((Source-Db-Table PATIENTS)
                (Source-Db-Column "doctor")
                (Source-Db-Datatype String)))

(defrelation ROOM_PATIENT.room
  :is-primitive DB-Relation
  :domain ROOM_PATIENT
  :annotations ((Source-Db-Table ROOM_PATIENT)
                (Source-Db-Column "room")
                (Source-Db-Datatype Number)))

(defrelation ROOM_PATIENT.patient
  :is-primitive DB-Relation
  :domain ROOM_PATIENT
  :annotations ((Source-Db-Table ROOM_PATIENT)
                (Source-Db-Column "patient")
                (Source-Db-Datatype String)))


(def-key-roles PATIENTS PATIENTS.patientid)
(def-key-roles ROOM_PATIENT ROOM_PATIENT.room)
```

The relational tables present in the databases have the following definitions:

```
*** In the "geo" database:

create table patients
(patientid  char(7) not null,
 lastname   char(15),
 firstname  char(15),
 dob        number,
 doctor     char(15)
);

SQL> describe patients
 Name                           Null?    Type
 ------------------------------ -------- ----
 PATIENTID                      NOT NULL CHAR(7)
 LASTNAME                                CHAR(15)
 FIRSTNAME                               CHAR(15)
 DOB                                     NUMBER
 DOCTOR                                  CHAR(15)

SQL> select patientid,lastname,firstname, dob, doctor from patients;

PATIENT LASTNAME        FIRSTNAME              DOB DOCTOR
------- --------------- --------------- ---------- ---------------
P1001   Okumura         Benjamin             20561 Fucich
P1002   DeSpain         Ann                 120442 Goldman
P1003   Kumar           Barbara              63065 Tzartzanis
P1004   Cooper          Albert              101030 Jain
P1005   Brown           Anant                30152 Gonzalez
P1006   Smith           Jacqueline           71570 Casner
P1007   Smith           Akitoshi             91851 Tzartzanis
P1008   Chame           Amanda               12745 Gonzalez
P1009   Kayano          Lee                 111740 Goldman
P1010   Hamilton        Sheila               30359 Hotz
P1011   Hammer          Janice               63077 Berson
P1012   Dosek           Thomas               41563 Woolf
P1013   Wills           Daniel               51772 Datta
P1014   Richardson      Vance                12268 Vernier
P1015   Mizushima       Dana                 40761 Gutierrez
```

```
*** In the "assets" database:

create table room_patient;
(room       number,
 patient    char(7)
);

SQL> describe room_patient;
 Name                           Null?    Type
 ------------------------------ -------- ----
 ROOM                           NOT NULL NUMBER
 PATIENT                                 CHAR(7)

SQL> select * from ROOM_PATIENT;

      ROOM PATIENT
---------- -------
       101 P1015
       102 P1002
       103 P1001
       104
       105 P1008
       106 P1011
       107
       108 P1014
       109 P1005
       110 P1007
       111 P1009
       112 P1013
       113 P1012
       114 P1004
       115
       116 P1010
       117
       118
       119 P1006
       120 P1003
```

10 Additional Reading

Using this manual and following the instructions in it require familiarity with SIMS, as well as with the Loom knowledge representation language, the LIM/IDI system for accessing remote information sources, and the KQML transport protocol.

The following papers may be consulted for further information about these programs.

10.1 SIMS

1. Arens, Y., Chee, C.Y., Hsu, C-N., and Knoblock, C.A. 1993. Retrieving and Integrating Data from Multiple Information Sources. In *International Journal of Intelligent and Cooperative Information Systems.* Vol. 2, No. 2. Pp. 127-158.

2. Arens, Y., Knoblock. C.A., and Shen W-M. Query Reformulation for Dynamic Information Integration, Submitted to *Journal of Intelligent Information Systems.*

3. Arens, Y. and Knoblock, C.A. 1994. Intelligent Caching: Selecting, Representing, and Reusing Data in an Information Server. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM-94)*, Gaithersburg, MD.

4. Arens, Y. and Knoblock, C.A. 1992. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems, *Proceedings of the First International Conference on Information and Knowledge Management (CIKM-92)*, Baltimore, MD.

5. Hsu, C-N., and Knoblock, C.A. 1995. Estimating the Robustness of Discovered Knowledge. in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, Montreal, Quebec, Canada.

6. Hsu, C-N., and Knoblock, C.A. 1995. Using inductive learning to gen- erate rules for semantic query optimization. In Gregory Piatetsky-Shapiro and Usama Fayyad, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 17. MIT Press.

7. Hsu, C-N., and Knoblock, C.A. 1994. Rule Induction for Semantic Query Optimization, in *Proceedings of the Eleventh International Conference on Machine Learning (ML-95)*, New Brunswick, NJ.

8. Hsu, C-N., and Knoblock, C.A. 1993. Reformulating Query Plans For Multidatabase Systems. In *Proceedings of the Second International Conference of Information and Knowledge Management (CIKM-93)*, Washington, D.C.

9. Knoblock, C.A., Arens, Y. and Hsu, C-N. 1994. An Architecture for Information Retrieval Agents. In *Proceedings of the Second International Conference on Cooperative Information Systems*, University of Toronto Publications, Toronto, Ontario, Canada.

10. Knoblock, C.A. 1995. Planning, Executing, Sensing, and Replanning for Information Gathering. In *IJCAI-95*, Montreal, Quebec, Canada.

11. Knoblock, C.A. 1994. Generating Parallel Execution Plans with a Partial-Order Planner. *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS94)*, Chicago, IL.

These publications, as well as additional information about SIMS, can be accessed through the WWW at `http://www.isi.edu/sims/`.

10.2 Loom

1. MacGregor, R. A Deductive Pattern Matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence.* St. Paul, MN, August 1988.

2. MacGregor, R. The Evolving Technology of Classification-Based Knowledge Representation Systems. In John Sowa (ed.), *Principles of Semantic Networks: Explorations in the Representation of Knowledge.* Morgan Kaufmann. 1990.

Additional papers and information about Loom can be accessed trough the WWW at the Loom Project homepage: `http://www.isi.edu/isd/LOOM/LOOM-HOME.html` .

10.3 LIM/IDI

1. McKay, D.P., Finin T., and O'Hare, A. The Intelligent Database Interface: Integrating AI and Database Systems. In *AAAI-90: Proceedings of The Eighth National Conference on Artificial Intelligence.* 1990.

2. Pastor, J. A., McKay. D.P., and Finin T. View-Concepts: KnowledgeBased Access to Databases. In *Proceedings of the First International Conference on Information and Knowledge Management,* Baltimore, MD. 1992.

3. LIM User's Manual. Available from Paramax Systems Corporation by anonymous FTP at `ftp://louise.vfl.paramax.com/`.

10.4 KQML

1. Finin, T., Fritzson, R. and McKay, D. A Language and Protocol to Support Intelligent Agent Interoperability. In *Proceedings of the CE and CALS Washington '92 Conference,* June, 1992.

Additional papers and information about KQML can be accessed through the WWW at the KQML homepage: `http://www.cs.umbc.edu/kqml/` .

Acknowledgements

We would like to thank the developers of the software systems that we have used extensively in the construction of SIMS. In particular, thanks to Bob Macgregor and Tom Russ for the Loom knowledge representation system. Thanks to Don Mckay, Jon Pastor, and Robin McEntire at Paramax/Unisys/Loral for both the LIM relational database wrapper and their implementation of the KQML language. And thanks to Dan Weld and Tony Barrett at the University of Washington for the UCPOP planner, which we used to build the SIMS planner. In addition, thanks to Ping Luo for his testing of and feedback on an earlier version of this manual.

References

[1] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[2] Tim Finin, Rich Fritzson, and Don McKay. A language and protocol to support intelligent agent interoperability. In *Proceedings of the CE and CALS*. Washington, D.C., June 1992.

[3] Robert MacGregor. A deductive pattern matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Saint Paul, Minnesota, 1988.

[4] Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1990.

[5] Donald P. McKay, Timothy W. Finin, and Anthony O'Hare. The intelligent database interface: Integrating AI and database systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. Boston, MA. 1990.

DISTRIBUTION LIST

addresses	number of copies
RAYMOND A. LIUZZI ROME LABORATORY/C3CA 525 BROOKS RD ROME NY 13441-4505	20
INFORMATION SCIENCES INSTITUTE UNIVERSITY OF SOUTHERN CALIFORNIA 4676 ADMIRALTY WAY MARINA DEL RAY CA 90292	5
ROME LABORATORY/SUL TECHNICAL LIBRARY 26 ELECTRONIC PKY ROME NY 13441-4514	1
ATTENTION: DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN ROAD, STE 0944 FT. BELVOIR, VA 22060-6218	2
ADVANCED RESEARCH PROJECTS AGENCY 3701 NORTH FAIRFAX DRIVE ARLINGTON VA 22203-1714	1
ROME LABORATORY/C3AB 525 BROOKS RD ROME NY 13441-4505	1
NAVAL WARFARE ASSESSMENT CENTER GIDEP (QA50) ATTN: RAYMOND TADROS PO BOX 8000 CORONA CA 91718-8000	1
WRIGHT LABORATORY/AAAI-2, BLDG 635 2185 AVIONICS CIRCLE WRIGHT-PATTERSON AFB OH 45433-7301	1

AFIT ACADEMIC LIBRARY/LDEE 1
2950 P STREET
AREA B, BLDG 642
WRIGHT-PATTERSON AFB OH 45433-7765

WRIGHT LABORATORY/MLPO 1
ATTN: R. L. DENISON
BLDG 651
3005 P STREET, STE 6
WRIGHT-PATTERSON AFB OH 45433-7707

AL/CFHI, BLDG 248 1
ATTN: GILBERT G. KUPERMAN
2255 H STREET
WRIGHT-PATTERSON AFB OH 45433-7022

AIR FORCE HUMAN RESOURCES LAB 1
TECHNICAL DOCUMENTS CENTER
OL AL HSC/HRG, BLDG 190
WRIGHT-PATTERSON AFB OH 45433-7604

AUL/LSE 1
BLDG 1405
600 CHENNAULT CIRCLE
MAXWELL AFB AL 361126424

US ARMY SPACE & STRATEGIC 1
DEFENSE COMMAND
CSSD-IM-PA
PO BOX 1500
HUNTSVILLE AL 35807-3801

NAVAL AIR WARFARE CENTER 1
6000 E. 21ST STREET
INDIANAPOLIS IN 46219-2189

COMMANDING OFFICER 1
NCCOSC RDTE DIVISION
ATTN: TECHNICAL LIBRARY,CODE 0274
53560 HULL STREET
SAN DIEGO CA 92152-5001

COMMANDER, TECHNICAL LIBRARY 1
4747000/C0223
NAVAIRWARCENWPNDIV
1 ADMINISTRATION CIRCLE
CHINA LAKE CA 93555-6001

SPACE & NAVAL WARFARE SYSTEMS
COMMAND, EXECUTIVE DIRECTOR (PD80A)
ATTN: MR. CARL ANDRIANI
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200 1

COMMANDER, SPACE & NAVAL WARFARE
SYSTEMS COMMAND (CODE 32)
2451 CRYSTAL DRIVE
ARLINGTON VA 22245-5200 1

US ARMY MISSILE COMMAND
AMSMI-RD-CS-R/DOCUMENTS
RSIC BLDG 4484
REDSTONE ARSENAL AL 35898-5241 2

LOS ALAMOS NATIONAL LABORATORY
PO BOX 1663
REPORT LIBRARY, P364
LOS ALAMOS NM 87545 1

AEDC LIBRARY
TECHNICAL REPORTS FILE
100 KINDEL DRIVE, SUITE C211
ARNOLD AFB TN 37389-3211 1

COMMANDER
USAISC
ASHC-IMD-L, BLDG 61801
FT HUACHUCA AZ 85613-5000 1

AFIWC/MSO
102 HALL BLVD, STE 315
SAN ANTONIO TX 78243-7016 1

DIRNSA
R509
9800 SAVAGE ROAD
FT MEADE MD 20755-6000 1

NSA/CSS
K1
FT MEADE MD 20755-6000 1

ESC/IC 1
50 GRIFFISS STREET
HANSCOM AFB MA 01731-1619

PHILLIPS LABORATORY 1
PL/TL (LIBRARY)
5 WRIGHT STREET
HANSCOM AFB MA 01731-3004

THE MITRE CORPORATION 1
ATTN: E. LADURE
D460
202 BURLINGTON RD
BEDFORD MA 01732

OUSD(P)/DTSA/DUTD 2
ATTN: PATRICK G. SULLIVAN, JR.
400 ARMY NAVY DRIVE
SUITE 300
ARLINGTON VA 22202

SOFTWARE ENGR'G INST TECH LIBRARY 1
ATTN: MR DENNIS SMITH
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213-3890

SOFTWARE OPTIONS, INC. 1
ATTN: MR TOM CHEATHAM
22 HILLIARD STREET
CAMBRIDGE MA 02138

USC-ISI 1
ATTN: DR ROBERT M. BALZER
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292-6695

KESTREL INSTITUTE 1
ATTN: DR CORDELL GREEN
1801 PAGE MILL ROAD
PALO ALTO CA 94304

ROCHESTER INSTITUTE OF TECHNOLOGY 1
ATTN: PROF J. A. LASKY
1 LOMB MEMORIAL DRIVE
P.O. BOX 9887
ROCHESTER NY 14613-5700

WESTINGHOUSE ELECTRONICS CORP 1
ATTN: MR DENNIS BIELAK
ELECTRONICS SYSTEMS GROUP
P.O. BOX 746, MAIL STOP 432
BALTIMORE MD 21203

AFIT/ENG 1
ATTN: PAUL BAILOR, MAJOR, USAF
WPAFB OH 45433-6583

THE MITRE CORPORATION 1
ATTN: MR EDWARD H. BENSLEY
BURLINGTON RD/MAIL STOP A350
BEDFORD MA 01730

ANDERSEN CONSULTING 1
ATTN: MR MICHAEL E. DEBELLIS
100 SOUTH WACKER DRIVE
CHICAGO IL 60606

UNIV OF ILLINOIS, URBANA-CHAMPAIGN 1
ATTN: DR MEHDI HARANDI
DEPT OF COMPUTER SCIENCES
1304 W. SPRINGFIELD/240 DIGITAL LAB
URBANA IL 61801

HONEYWELL, INC. 1
ATTN: MR BERT HARRIS
FEDERAL SYSTEMS
7900 WESTPARK DRIVE
MCLEAN VA 22102

SOFTWARE ENGINEERING INSTITUTE 1
ATTN: MR WILLIAM E. HEFLEY
CARNEGIE-MELLON UNIVERSITY
SEI 2218
PITTSBURGH PA 15213-38990

UNIVERSITY OF SOUTHERN CALIFORNIA 1
ATTN: DR W. LEWIS JOHNSON
INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY/SUITE 1001
MARINA DEL REY CA 90292-6695

COLUMBIA UNIV/DEPT COMPUTER SCIENCE 1
ATTN: DR GAIL E. KAISER
450 COMPUTER SCIENCE BLDG
500 WEST 120TH STREET
NEW YORK NY 10027

SOFTWARE PRODUCTIVITY CONSORTIUM 1
ATTN: MR ROBERT LAI
2214 ROCK HILL ROAD
HERNDON VA 22070

AFIT/ENG 1
ATTN: DR GARY B. LAMONT
SCHOOL OF ENGINEERING
DEPT ELECTRICAL & COMPUTER ENGRG
WPAFB OH 45433-6583

NSA/OFC OF RESEARCH 1
ATTN: MS MARY ANNE OVERMAN
9800 SAVAGE ROAD
FT GEORGE G. MEADE MD 20755-6000

AT&T BELL LABORATORIES 1
ATTN: MR PETER G. SELFRIDGE
ROOM 3C-441
600 MOUNTAIN AVE
MURRAY HILL NJ 07974

VITRO CORPORATION 1
ATTN: MR ROBERT A. SMALL
14000 GEORGIA AVENUE
SILVER SPRING MD 20906-2972

ODYSSEY RESEARCH ASSOCIATES, INC. 1
ATTN: MS MAUREEN STILLMAN
301A HARRIS B. DATES DRIVE
ITHACA NY 14850-1313

WRDC/AAAF-3 1
ATTN: JAMES P. WEBER, CAPT, USAF
AERONAUTICAL SYSTEMS CENTER
WPAFB OH 45433-6543

TEXAS INSTRUMENTS INCORPORATED 1
ATTN: DR DAVID L. WELLS
P.O. BOX 655474, MS 238
DALLAS TX 75265

BOEING COMPUTER SERVICES 1
ATTN: DR PHIL NEWCOMB
MS 7L-64
P.O. BOX 24346
SEATTLE WA 98124-0346

LOCKHEED SOFTWARE TEHNOLOGY CENTER 1
ATTN: MR HENSON GRAVES
ORG. 96-LO BLOG 254E
3251 HANOVER STREET
PALO ALTO CA 94304-LL9L

REASONING SYSTEMS 1
ATTN: DR GORDON KOTIK
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

TEXAS A & M UNIVERSITY 1
ATTN: DR PAULA MAYER
KNOWLEDGE BASED SYSTEMS LABORATORY
DEPT OF INDUSTRIAL ENGINEERING
COLLEGE STATION TX 77843

KESTREL DEVELOPMENT CORPORATION 1
ATTN: DR RICHARD JULLIG
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

ARPA/SISTO 1
ATTN: DR KIRSTIE BELLMAN
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

LOCKHEED O/96-10 B/254E 1
ATTN: JACKY COMBS
3251 HANOVER STREET
PALO ALTO CA 94304-1191

NASA/JOHNSON SPACE CENTER 1
ATTN: CHRIS CULBERT
MAIL CODE PT4
HOUSTON TX 77058

SAIC 1
ATTN: LANCE MILLER
MS T1-6-3
PO BOX 1303 (OR 1710 GOODRIDGE DR)
MCLEAN VA 22102

STERLING IMD INC. 1
KSC OPERATIONS
ATTN: MARK MAGINN
BEECHES TECHNICAL CAMPUS/RT 26 N.
ROME NY 13440

NAVAL POSTGRADUATE SCHOOL 1
ATTN: BALA RAMESH
CODE AS/RS
ADMINISTRATIVE SCIENCES DEPT
MONTEREY CA 93943

KESTREL INSTITUTE 1
ATTN: MARIA PRYCE
3260 HILLVIEW AVENUE
PALO ALTO CA 94304

HUGHES AIRCRAFT COMPANY 1
ATTN: GERRY BARKSDALE
P. O. BOX 3310
BLDG 618 MS E215
FULLERTON CA 92634

SCHLUMBERGER LABORATORY FOR 1
COMPUTER SCIENCE
ATTN: DR. GUILLERMO ARANGO
8311 NORTH FM620
AUSTIN, TX 78720

PARAMAX SYSTEMS CORPORATION 1
ATTN: DON YU
8201 GREENSBORO DRIVE, SUITE 1000
MCLEAN VA 22101

MOTOROLA, INC. 1
ATTN: MR. ARNOLD PITTLER
3701 ALGONQUIN ROAD, SUTE 601
ROLLING MEADOWS, IL 60008

DECISION SYSTEMS DEPARTMENT 1
ATTN: PROF WALT SCACCHI
SCHOOL OF BUSINESS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CA 90089-1421

SOUTHWEST RESEARCH INSTITUTE 1
ATTN: BRUCE REYNOLDS
6220 CULEBRA ROAD
SAN ANTONIO, TX 78228-0510

NATIONAL INSTITUTE OF STANDARDS 1
AND TECHNOLOGY
ATTN: CHRIS DABROWSKI
ROOM A266, BLDG 225
GAITHSBURG MD 20899

EXPERT SYSTEMS LABORATORY 1
ATTN: STEVEN H. SCHWARTZ
NYNEX SCIENCE & TECHNOLOGY
500 WESTCHESTER AVENUE
WHITE PLANS NY 20604

NAVAL TRAINING SYSTEMS CENTER 1
ATTN: ROBERT BREAUX/CODE 252
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3224

CENTER FOR EXCELLENCE IN COMPUTER- 1
AIDED SYSTEMS ENGINEERING
ATTN: PERRY ALEXANDER
2291 IRVING HILL ROAD
LAWRENCE KS 66049

SOFTWARE TECHNOLOGY SUPPORT CENTER 1
ATTN: MAJ ALAN K. MILLER
OGDEN ALC/TISE
BLDG 100, BAY G
HILL AFB, UTAH 84056

DR JAMES ALLEN 1
COMPUTER SCIENCE DEPT/BLDG RM 732
UNIV OF ROCHESTER
WILSON BLVD
ROCHESTER NY 14627

DR YIGAL ARENS 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR RAY BAREISS 1
THE INST. FOR LEARNING SCIENCES
NORTHWESTERN UNIV
1890 MAPLE AVE
EVANSTON IL 60201

MR. JEFF BERLINER 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR MARIE A. BIENKOWSKI 1
SRI INTERNATIONAL
333 RAVENSWOOD AVE/EK 337
MENLO PRK CA 94025

DR MARK S. BODDY 1
HONEYWELL SYSTEMS & RSCH CENTER
3660 TECHNOLOGY DRIVE
MINNEAPOLIS MN 55418

DR PIERO P. BONISSONE 1
GE CORPORATE RESEARCH & DEVELOPMENT
BLDG K1-RM 5C-32A
P. O. BOX 8
SCHENECTADY NY 12301

MR. DAVID BROWN 1
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

DR MARK BURSTEIN 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR GREGG COLLINS 1
INST FOR LEARNING SCIENCES
1890 MAPLE AVE
EVANSTON IL 60201

MR. RANDALL J. CALISTRI-YEH 1
ORA CORPORATION
301 DATES DRIVE
ITHACA NY 14850-1313

DR STEPHEN E. CROSS 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213

DR THOMAS CHEATHAM 1
HARVARD UNIVERSITY
DIV OF APPLIED SCIENCE
AIKEN, RM 104
CAMBRIDGE MA 02138

MS. LAURA DAVIS 1
CODE 5510
NAVY CTR FOR APPLIED RES IN AI
NAVAL RESEARCH LABORATORY
WASH DC 20375-5337

MS. GLADYS CHOW 1
COMPUTER SCIENCE DEPT.
UNIV OF CALIFORNIA
LOS ANGELES CA 90024

DR THOMAS L. DEAN 1
BROWN UNIVERSITY
DEPT OF COMPUTER SCIENCE
P.O. BOX 1910
PROVIDENCE RI 02912

DR WESLEY CHU 1
COMPUTER SCIENCE DEPT
UNIV OF CALIFORNIA
LOS ANGELES CA 90024

MR. ROBERTO DESIMONE 1
SRI INTERNATIONAL (EK335)
333 RAVENSWOOD AVE
MENLO PRK CA 94025

DR PAUL R. COHEN 1
UNIV OF MASSACHUSETTS
COINS DEPT
LEDERLE GRC
AMHERST MA 01003

DR JON DOYLE 1
LABORATORY FOR COMPUTER SCIENCE
MASS INSTITUTE OF TECHNOLOGY
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

DR. BRIAN DRABBLE 1
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH/80 S. BRIDGE
EDINBURGH EH1 LHN
UNITED KINGDOM

MR. SCOTT FOUSE 1
ISX CORPORATION
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

MR. STU DRAPER 1
MITRE
EAGLE CENTER 3, SUITE 8
O'FALLON IL 62269

DR MARK FOX 1
DEPT O INDUSTRIAL ENGRG
UNIV OF TORONTO
4 TADDLE CREAK ROAD
TORONTO, ONTARIO, CANADA

MR. GARY EDWARDS 1
4353 PARK TERRACE DRIVE
WESTLAKE VILLACA 91361

MR. RUSS FREW 1
GENERAL ELECTRIC
MOORESTOWN CORPORATE CENTER
BLDG ATK 145-2
MOORESTOWN NJ 08057

DR MICHAEL FEHLING 1
STANFORD UNIVERSITY
ENGINEERING ECO SYSTEMS
STANFORD CA 94305

MR. RICH FRITZSON 1
CENTER OR ADVANCED INFO TECHNOLOGY
UNISYS
P.O. BOX 517
PAOLI PA 19301

DR KRISTIAN J. HAMMOND 1
UNIV OF CHICAGO
COMPUTER SCIENCE DEPT/RY155
1100 E. 58TH STREET
CHICAGO IL 60637

RICK HAYES-ROTH 1
CIMFLEX-TEKNOWLEDGE
1810 EMBARCADERO RD
PALO ALTO CA 94303

DR JIM HENDLER 1
UNIV OF MARYLAND
DEPT OF COMPUTER SCIENCE
COLLEGE PARK MD 20742

MS. YOLANDA GIL 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR MAX HERION 1
ROCKWELL INTERNATIONAL SCIENCE CTR
444 HIGH STREET
PALO ALTO CA 94301

MR. MORTON A. HIRSCHBERG, DIRECTOR 1
US ARMY RESEARCH LABORATORY
ATTN: AMSRL-CI-CB
ABERDEEN PROVING GROUND MD
21005-5066

MR. MARK A. HOFFMAN 1
ISX CORPORATION
1165 NORTHCHASE PARKWAY
MARIETTA GA 30067

DR RON LARSEN 1
NAVAL CMD, CONTROL & OCEAN SUR CTR
RESEARCH, DEVELOP, TEST & EVAL DIV
CODE 444
SAN DIEGO CA 92152-5000

DR CRAIG KNOBLOCK 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

MR. RICHARD LOWE (AP-10) 1
SRA CORPORATION
2000 15TH STREET NORTH
ARLINGTON VA 22201

MR. TED C. KRAL 1
BBN SYSTEMS & TECHNOLOGIES
4015 HANCOCK STREET, SUITE 101
SAN DIEGO CA 92110

DR JOHN LOWRENCE 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. ALAN MEYROWITZ 1
NAVAL RESEARCH LABORATORY/CODE 5510
4555 OVERLOOK AVE
WASH DC 20375

ALICE MULVEHILL 1
MITRE CORPORATION
BURLINGTON RD
M/S K-302
BEDFORD MA 01730

DR ROBERT MACGREGOR 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL REY CA 90292

DR DREW MCDERMOTT 1
YALE COMPUTER SCIENCE DEPT
P.O. BOX 2158, YALE STATION
51 PROPSPECT STREET
MEW HAVEN CT 06520

MS. CECILE PARIS 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR DOUGLAS SMITH 1
KESTREL INSTITUTE
3260 HILLVIEW AVE
PALO ALTO CA 94304

DR. AUSTIN TATE 1
AI APPLICATIONS INSTITUTE
UNIV OF EDINBURGH
80 SOUTH BRIDGE
EDINBURGH EH1 1HN - SCOTLAND

DIRECTOR 1
DARPA/ITO
3701 N. FAIRFAX DR., 7TH FL
ARLINGTON VA 22209-1714

DR STEPHEN F. SMITH 1
ROBOTICS INSTITUTE/CMU
SCHENLEY PRK
PITTSBURGH PA 15213

DR. ABRAHAM WAKSMAN 1
AFOSR/NM
110 DUNCAN AVE., SUITE B115
BOLLING AFB DC 20331-0001

DR JONATHAN P. STILLMAN 1
GENERAL ELECTRIC CRD
1 RIVER RD, RM K1-5C31A
P. O. BOX 8
SCHENECTADY NY 12345

DR EDWARD C.T. WALKER 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

DR BILL SWARTOUT 1
USC/ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

GIO WIEDERHOLD 1
STANFORD UNIVERSITY
DEPT OF COMPUTER SCIENCE
438 MARGARET JACKS HALL
STANFORD CA 94305-2140

DR KATIA SYCARA/THE ROBOTICS INST 1
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIV
DOHERTY HALL RM 3325
PITTSBURGH PA 15213

DR DAVID E. WILKINS 1
SRI INTERNATIONAL
ARTIFICIAL INTELLIGENCE CENTER
333 RAVENSWOOD AVE
MENLO PARK CA 94025

DR. PATRICK WINSTON 1
MASS INSTITUTE OF TECHNOLOGY
RM NE43-817
545 TECHNOLOGY SQUARE
CAMBRIDGE MA 02139

HUA YANG 1
COMPUTER SCIENCE DEPT
UNIV OF CALIORNIA
LOS ANGELES CA 90024

MR. RICK SCHANTZ 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON STREET
CAMBRIDGE MA 02138

MR JOHN P. SCHILL 1
ARPA/ISO
3701 N FAIRFAX DRIVE
ARLINGTON VA 22203-1714

DR STEVE ROTH 1
CENTER FOR INTEGRATED MANUFACTURING
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIV
PITTSBURGH PA 15213-3890

DR YOAV SHOHAM 1
STANFORD UNIVERSITY
COMPUTER SCIENCE DEPT
STANFORD CA 94305

MR. MIKE ROUSE 1
AFSC
7800 HAMPTON RD
NORFOLK VA 23511-6097

MR. DAVID E. SMITH 1
ROCKWELL INTERNATIONAL
444 HIGH STREET
PALO ALTO CA 94301

JEFF ROTHENBERG 1
SENIOR COMPUTER SCIENTIST
THE RAND CORPORATION
1700 MIN STREET
SANTA MONICA CA 90407-2138

DR LARRY BIRNBAUM 1
NORTHWESTERN UNIVERSITY
ILS
1890 MAPLE AVE
EVANSTON IL 60201

MR. LEE ERMAN 1
CIMFLEX TECKNOWLEDGE
1810 EMBARCARDERO RD
PALO ALTO CA 94303

MR DICK ESTRADA 1
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON ST
CAMBRIDGE MA 02138

MR HARRY FORSDICK 1
BBN SYSTEMS AND TECHNOLOGIES
10 MOULTON ST
CAMBRIDGE MA 02138

DR MATTHEW L. GINSBERG 5
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403

MR IRA GOLDSTEIN 1
OPEN SW FOUNDATION RESEARCH INST
ONE CAMBRIDGE CENTER
CAMBRIDGE MA 02142

DR MOISES GOLDSZMIDT 1
INFORMATION AND DECISION SCIENCES
ROCKWELL INTL SCIENCE CENTER
444 HIGH ST, SUITE 400
PALO ALTO CA 94301

MR JEFF GROSSMAN, CO 1
NCCOSC RDTE DIV 44
5370 SILVERGATE AVE, ROOM 1405
SAN DIEGO CA 92152-5146

JAN GUNTHER 1
ASCENT TECHNOLOGY, INC.
64 SIDNEY ST, SUITE 380
CAMBRIDGE MA 02139

DR LYNETTE HIRSCHMAN 1
MITRE CORPORATION
202 BURLINGTON RD
BEDFORD MA 01730

DR ADELE E. HOWE 1
COMPUTER SCIENCE DEPT
COLORADO STATE UNIVERSITY
FORT COLLINS CO 80523

DR LESLIE PACK KAELBLING 1
COMPUTER SCIENCE DEPT
BROWN UNIVERSITY
PROVIDENCE RI 02912

DR SUBBARAO KAMBHAMPATI 1
DEPT OF COMPUTER SCIENCE
ARIZONA STATE UNIVERSITY
TEMPE AZ 85287-5406

MR THOMAS E. KAZMIERCZAK 1
SRA CORPORATION
331 SALEM PLACE, SUITE 200
FAIRVIEW HEIGHTS IL 62208

DR PRADEEP K. KHOSLA 1
ARPA/ITO
3701 N. FAIRFAX DR
ARLINGTON VA 22203

DR CRAIG KNOBLOCK 1
USC-ISI
4676 ADMIRALTY WAY
MARINA DEL RAY CA 90292

DR CARLA GOMES 1
ROME LABORATORY/C3CA
525 BROOKS RD
ROME NY 13441-4505

DR MARK T. MAYBURY 1
ASSOCIATE DIRECTOR OF AI CENTER
ADVANCED INFO SYSTEMS TECH G041
MITRE CORP, BURLINGTON RD, MS K-329
BEDFORD MA 01730

MR DONALD P. MCKAY 1
PARAMAX/UNISYS
P O BOX 517
PAOLI PA 19301

DR KAREN MYERS 1
AI CENTER
SRI INTERNTIONAL
333 RAVENSWOOD
MENLO PARK CA 94025

DR MARTHA E POLLACK 1
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF PITTSBURGH
PITTSBURGH PA 15260

DR RAJ REDDY
SCHOOL OF COMPUTER SCIENCE
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PA 15213 1

DR EDWINA RISSLAND
DEPT OF COMPUTER & INFO SCIENCE
UNIVERSITY OF MASSACHUSETTS
AMHERST MA 01003 1

DR MANUELA VELOSO
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891 1

DR DAN WELD
DEPT OF COMPUTER SCIENCE & ENG
MAIL STOP FR-35
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195 1

MR JOE ROBERTS
ISX CORPORATION
4301 N FAIRFAX DRIVE, SUITE 301
ARLINGTON VA 22203 1

COL JOHN A. WARDEN III
ASC/CC
225 CHENNAULT CIRCLE
MAXWELL AFB AL 36112-6426 1

DR TOM GARVEY
ARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

MR JOHN N. ENTZMINGER, JR.
ARPA/DIRO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

LT COL ANTHONY WAISANEN, PHD
COMMAND ANALYSIS GROUP
HQ AIR MOBILITY COMMAND
402 SCOTT DRIVE, UNIT 3L3
SCOTT AFB IL 62225-5307 1

DIRECTOR
ARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

OFFICE OF THE CHIEF OF NAVAL RSCH
ATTN: MR PAUL QUINN
CODE 311
800 N. QUINCY STREET
ARLINGTON VA 22217 1

BBN SYSTEMS AND TECHNOLOGY
ATTN: MR MAURICE MCNEIL
9655 GRANITE RIDGE DRIVE, SUITE 245
SAN DIEGO CA 92123 1

DR JOHN SALASIN
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

DR HOWIE SHROBE
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

DR HOWARD FRANK
DARPA/ITO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714 1

DR LARRY DRUFFLE
SOFTWARE ENGINEERING INSTITUTE
CARNEGIE MELLON UNIV, SCHENLEY PK
4500 FIFTH AVE, ROOM 2204
PITTSBURGH PA 15213 1

DR BARRY BOEHM
DIR, USC CENTER FOR SW ENGINEERING
COMPUTER SCIENCE DEPT
UNIV OF SOUTHERN CALIFORNIA
LOS ANGELES CA 90089-0781 1

DR STEVE CROSS
CARNEGIE MELLON UNIVERSITY
SCHOOL OF COMPUTER SCIENCE
PITTSBURGH PA 15213-3891 1

DR MARK MAYBURY 1
MITRE CORPORATION
ADVANCED INFO SYS TECH; G041
BURLINTON ROAD, M/S K-329
BEDFORD MA 01730

MR SCOTT FOUSE 1
ISX
4353 PARK TERRACE DRIVE
WESTLAKE VILLAGE C 91361

MR GARY EDWARDS 1
ISX
433 PARK TERRACE DRIVE
WESTLAKE VILLAGE CA 91361

DR ED WALKER 1
BBN SYSTEMS & TECH CORPORATION
10 MOULTON STREET
CAMBRIDGE MA 02238

DR EDWARD FEIGENBAUM 1
KOWLEDGE SYSTEMS LAB
STANFORD UNIVERSITY
701 WELCH ROAD, BUILDING C
PALO ALTO CA 94304

JEFFREY D. GRIMSHAW, CAPT, USAF 1
HQ USAFA/DFCS
2354 FAIRCHILD DRIVE, SUITE 6K41
USAF ACADEMY
COLORDO SPRINGS CO 80840-6234

LEE ERMAN 1
CIMFLEX TEKNOWLEDGE
1810 EMBACADERO ROAD
P.O. BOX 10119
PALO ALTO CA 94303

DR OREN ETZIONI 1
DEPT OF COMPUTER SCIENCE
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

DR GEORGE ERGUSON 1
UNIVERSITY OF ROCHESTER
COMPUTER STUDIES BLDG, RM 732
WILSON BLVD
ROCHESTER NY 14627

DR STEVE HANKS
DEPT OF COMPUTER SCIENCE & ENG'G
UNIVERSITY OF WASHINGTON
SEATTLE WA 98195

1

DR WILLIAM S. MARK
LOCKHEED PALO ALTO RSCH LAB
DEPT 9620, BLDG 254F
3251 HANOVER ST
PALO ALTO CA 94304-1187

1

MR DON MORROW
BBN SYSTEMS & TECHOLOGIES
101 MOONGLOW DR
BELLEVILLE IL 62221

1

DR KAREN MYERS
AI CENTER
SRI INTERNATIONAL
333 RAVENSWOOD
MENLO PARK CA 94025

1

DR CHRISTOPHER OWENS
BBN SYSTEMS & TECHNOLOGIES
10 MOULTON ST
CAMBRIDGE MA 02138

1

DR ADNAN DARWICHE
INFORMATION & DECISION SCIENCES
ROCKWELL INT'L SCIENCE CENTER
1049 CAMINO DOS RIOS
THOUSAND OAKS CA 91360

1

DR JAIME CARBONNEL
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3325
PITTSBURGH PA 15213

1

NR NORMAN SADEH
THE ROBOTICS INSTITUTE
CARNEGIE MELLON UNIVERSITY
DOHERTY HALL, ROOM 3315
PITTSBURGH PA 15213

1

DR JAMES CRAWFORD
CIRL, 1269
UNIVERSITY OF OREGON
EUGENE OR 97403

DR TAIEB ZNATI 1
UNIVERSITY OF PITTSBURGH
DEPT OF COMPUTER SCIENCE
PITTSBURGH PA 15260

DR MARIE DEJARDINS 1
SRI INTERNATIONAL
333 RAVENSWOOD AVENUE
MENLO PARK CA 94025

ROBERT J. KRUCHTEN 1
HQ AMC/SCA
203 W LOSEY ST, SUITE 1016
SCOTT AFB IL 62225-5223

DR. DAVE GUNNING 1
DARPA/ISO
3701 NORTH FAIRFAX DRIVE
ARLINGTON VA 22203-1714

MS. LEAH WONG 1
NCCOSC RDTE DIVISIION
53560 HULL STREET
SAN DIEGO CA 92152-5001

*U.S. GOVERNMENT PRINTING OFFICE: 1996-710-126-47034

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.